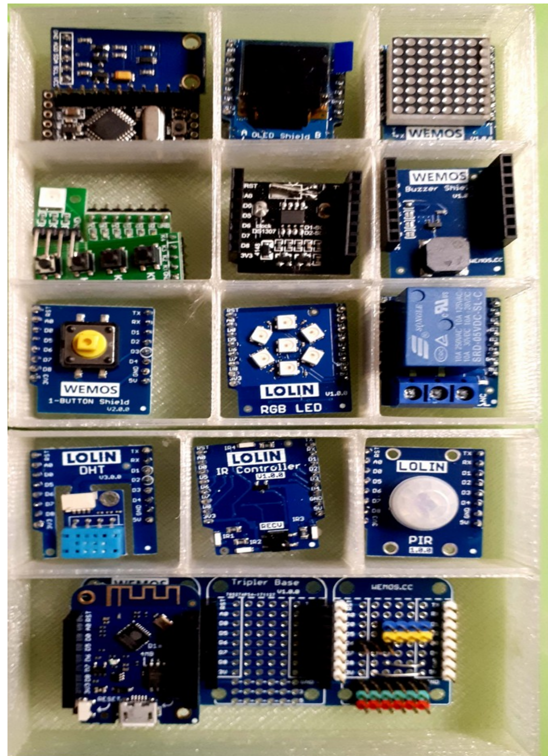
IoT amb D1 mini (ESP8266) i codi Arduino



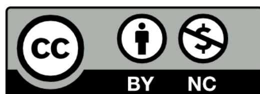
Jordi Orts

Gener 2019

IoT amb D1 mini (ESP8266) i codi Arduino

Jordi Orts

1a edició, 1a revisió (12/01/2018)
Gener 2019



Trobareu la versió més recent d'aquest llibre a https://github.com/jorts64/kit-D1-mini/blob/master/IoT-D1-1a-ed.zip

Trobareu tots els exemples del llibre i més a https://github.com/jorts64/kit-D1-mini/

Taula de continguts

Agraïments

Vull agrair a tots els alumnes que han fet amb mi el seu Treball Final de Màster, Treball de Recerca de Batxillerat, Projecte de Recerca de 4t d'ESO, o projectes al taller de Tecnologia. Gràcies a ells la meva experiència ha crescut al llarg dels anys de forma exponencial. Gràcies de tot cor.

Entre aquests antics alumnes, vull dedicar especialment aquest llibre a la meva filla Ingrid i al meu nebot Javi. El tracte continu que tinc amb ells em permet veure, en els seus projectes, en la seva forma de treballar, com la llavor que vaig plantar en ells fa anys ha crescut, confirmant el que jo pensava i enfortint les meves creences sobre pedagogia STEAM. Una braçada per tots dos.

També vull agrair al grup de treball de professors sobre D1 mini i IoT els seus ànims i suport. Sense ells no hauria estat possible escriure aquest llibre en menys d'un mes. Espero que aquest llibre ens serveixi de guia per desenvolupar nous projectes i assolir noves fites. Endavant!

Jordi Orts

5 de gener de 2019

Pròleg

Introducció

Aquest llibre està destinat a tots aquells que, tenint un coneixement bàsic de programació en Arduino[1] i el seu IDE, volen treballar amb els ESP8266[2] en aquest entorn.

Porto treballant amb l’ESP8266 des de març del 2016. Es a dir, gairebé 3 anys. Si bé a l’estiu d’aquell any vaig explorar intensament les plaques NodeMCU, i vaig quedar meravellat, especialment amb la utilització de l’arduino IDE per a aquestes plaques , vaig descobrir tot el potencial d’aquestes joies un parell de mesos després, amb el sistema modular D1 mini.

Aquest sistema reuneix tres avantatges importants:

- Es treballa en un entorn conegut, l’arduino IDE, de forma que els alumnes poden fer una transició suau des de la programació tradicional amb arduino. L’aparició en aquest entorn de la gestió de targetes externes amb la versió 1.6.4 de l’IDE va facilitar molt l’adaptació d’aquest.
- Aquesta adaptació de l’IDE d’arduino introdueix noves prestacions com la interfície WiFi integrada, la creació de SDs virtuals amb el sistema de fitxers SPIFFS o la generació de freqüències variables de sortida. Realment s’ha fet un gran treball en aquest sentit.
- El sistema de shields del D1 mini permet un disseny realment modular. Amb un arduino tradicional podem tenir problemes

1 La programació de l’ESP8266 no està limitada a l’IDE Arduino. Podeu veure altres llenguatges de programació admesos per aquest xip a Annex 2: Altres llenguatges de programació per l’ESP8266

2 Podeu veure més informació sobre l’ESP826 a Annex 1: Microcontrolador ESP8266

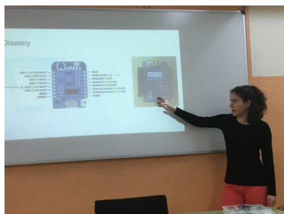
per treballar amb 2 shields i algun component extern. Amb el D1 mini podem treballar fàcilment amb 4 o 5 shields i afegir components externs. I a un preu molt raonable.

Un any després els meus alumnes de 4t d'ESO ja utilitzaven aquests mòduls en els seus projectes de recerca, amb resultats sorprenents. Al curs següent vaig dissenyar la primera versió del kit que aquí us presentem, gràcies a l'aparició del primer distribuïdor a Espanya d'aquests mòduls. Amb aquest kit es va treballar a 1r de Batxillerat, amb un rotund èxit al Maker Faire BNC 2018, on els meus alumnes van ser premiats amb un *Maker of Merit*[IPV01].

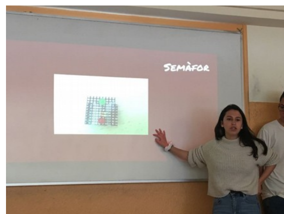
Al setembre de 2018 el meu nebot Javi Orts i jo vam dissenyar i fabricar el nostre primer shield pel D1 mini.

A la jornada «Engrescant el jovent cap a la tecno»[CES01] que es va realitzar a la ESEIAAT-UPC de Terrassa l'1 de desembre de 2018 vaig tenir l'ocasió de presentar en un taller el kit, amb gran expectació i satisfacció per part dels professors assistents (va haver overbooking al meu taller!). I allà mateix va sortir la idea de crear un grup de treball de professors per aprofundir i difondre el tema.

En tres anys vaig fer innombrables proves, vaig tutoritzar diversos PREs, TRs, TFMs, projectes al taller de Batxillerat ... Aprenent cada vegada més amb els meus alumnes[3].







3 Podeu veure una selecció d'aquests treballs a https://github.com/jorts64/kit-D1-mini/tree/master/projectes

Naturalment, ja em coneixeu: calia fer un llibre per transmetre tots els coneixements adquirits al llarg d'aquest temps, a la vegada que ordenava les meves idees. I aquí el teniu!

Jordi Orts

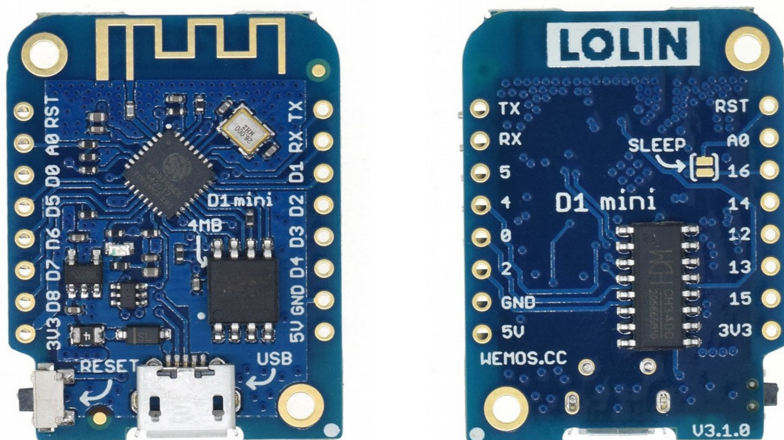
Barcelona, 5 de gener de 2019

El kit D1 mini R1

Fruit dels resultats de l'anterior kit [GIT01], que em va permetre reduir el temps de muntatge de prototips al taller considerablement, i amb la intenció de guanyar compatibilitat[4] entre els mòduls i millorar les prestacions del kit, a l'octubre del 2018 vaig dissenyar una revisió[GIT02] d'aquest on no em vaig limitar als productes disponibles al distribuïdor espanyol.

Amb aquest nou conjunt de materials és poden fer fàcilment multitud de prototips comercials.

Aquest kit, com el seu nom indica, està dissenyat al voltant del D1 mini, una placa amb un ESP8266 amb 4MB de memòria[5], antena WiFi integrada i un bus que conté tots els senyals aprofitables de l'ESP8266.

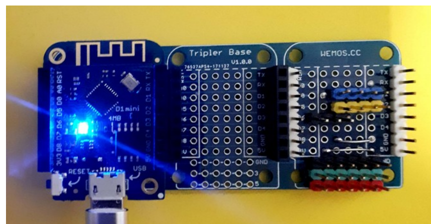


4 Veure Annex 3: Connexions i compatibilitat dels shields

5 Si necessiteu més memòria, o una antena WiFi més potent, podeu fer servir el D1 mini pro de 16 MB. Més informació a Annex 4: D1 mini pro

Contingut del kit

Tripler base amb D1 mini, terminals mascle a la tercera columna per I/O i alimentació (GND-3,3V-5V), connector 3 pins sortida de servo (connectat a D6) a 5V i connexions I2C



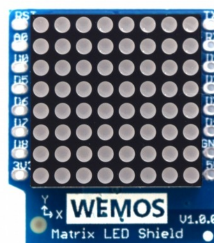
OLED Shield V2.1.0

Terminals M



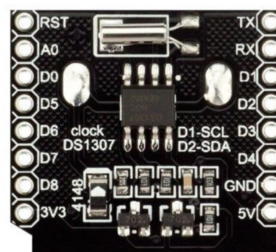
Matrix led shield

Terminals M



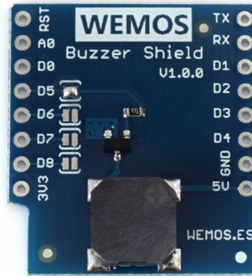
RTC shield

Terminals M-F



Buzzer shield

Terminals M-F



1-button shield

Terminals M



RGB shield

Terminals M

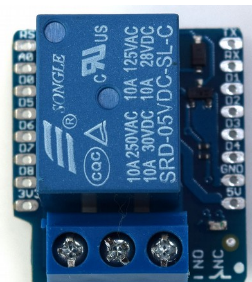
Reconfigurat al pin D8



Relay shield V2.0.0

Terminals M

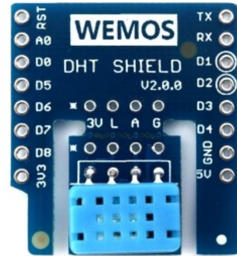
Reconfigurat al pin D7



El kit D1 mini R1

DHT shield

Terminals M-F



IR controller Shield

Terminals M



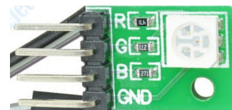
PIR shield

Terminals M

Reconfigurat al pin D6



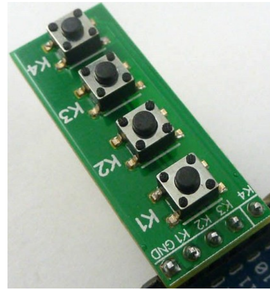
Mòdul LED RGB



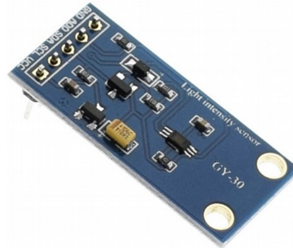
Mòdul 6 LEDs



Mòdul 4 polsadors



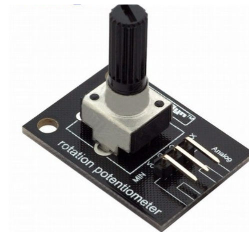
Mòdul sensor de llum I2C BH1750FVI



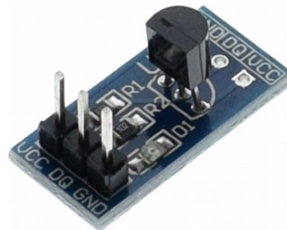
Mòdul acceleròmetre + giroscopi I2C GY-521



Mòdul potenciòmetre



Mòdul sensor temperatura DS18B20

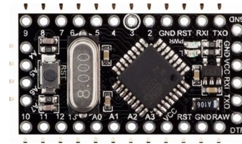


El kit D1 mini R1

Microservo 3,7g



Arduino Pro mini com a esclau I2C



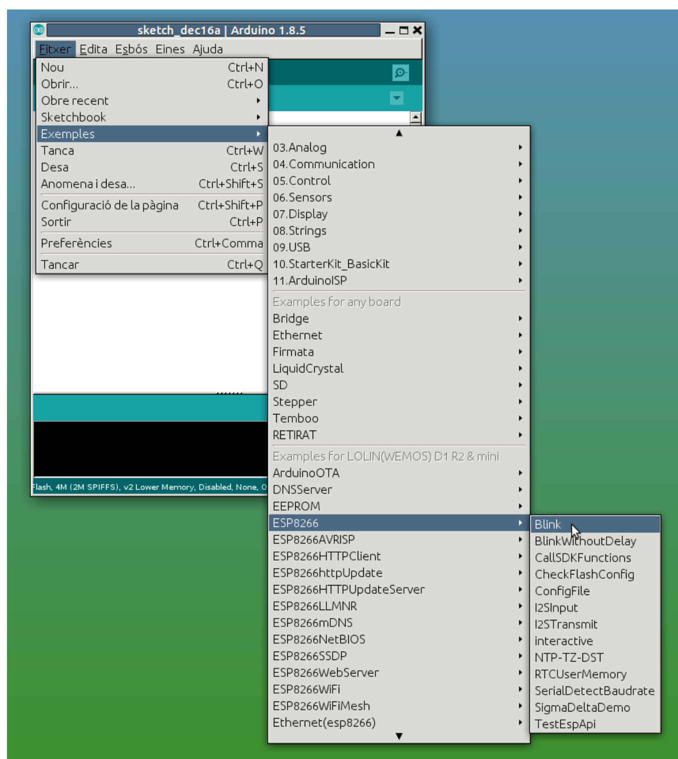
10 cables Dupont F-F



Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

És típic quan un comença a treballar amb Arduino carregar l'exemple *Blink*, que fa pampallugues al led integrat en la placa Arduino.

Nosaltres podem començar igual amb el D1 mini. Però haurem de fer servir l'exemple creat especialment pel ESP8266:



```
/*
  ESP8266 Blink by Simon Peter
  Blink the blue LED on the ESP-01 module
  This example code is in the public domain

  The blue LED on the ESP-01 module is connected to GPIO1
  (which is also the TXD pin; so we cannot use Serial.print() at the same time)

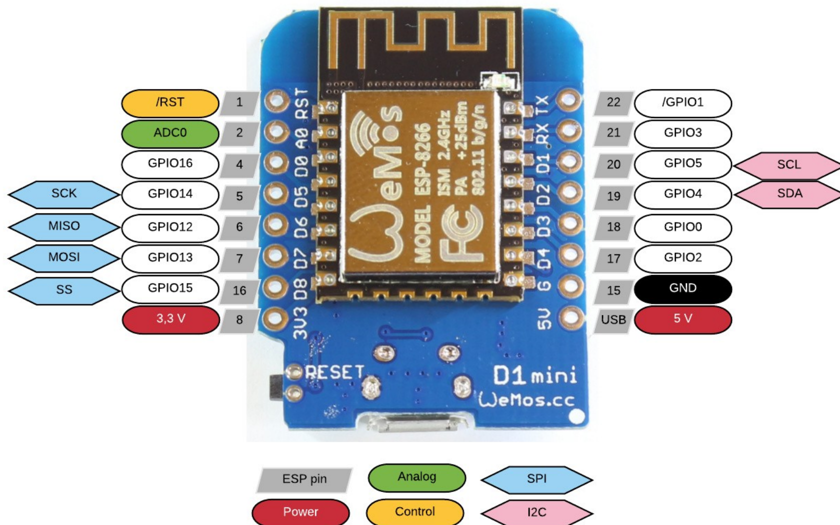
  Note that this sketch uses LED_BUILTIN to find the pin with the internal LED
*/

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);     // Initialize the LED_BUILTIN pin as an output
}

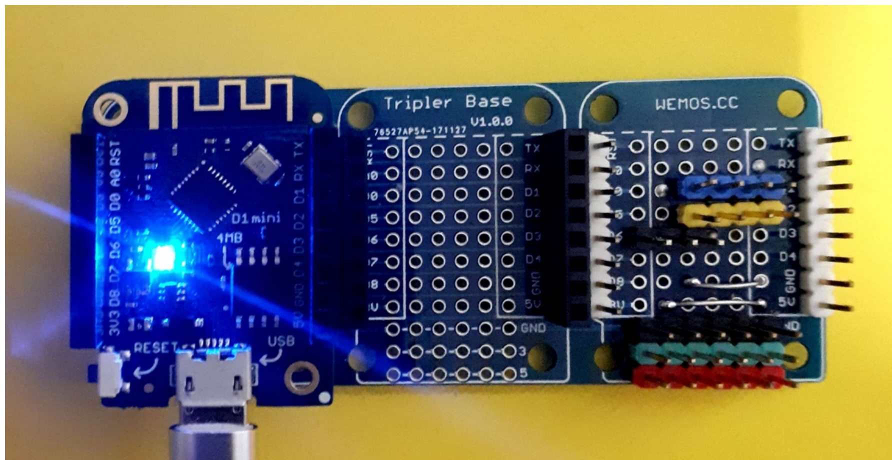
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, LOW);   // Turn the LED on (Note that LOW is the voltage level
  // but actually the LED is on; this is because
  // it is active low on the ESP-01)
  delay(1000);                      // Wait for a second
  digitalWrite(LED_BUILTIN, HIGH);  // Turn the LED off by making the voltage HIGH
  delay(2000);                      // Wait for two seconds (to demonstrate the active low LED)
}
```

Fixeu-vos com al programa s'utilitza el pin *LED_BUILTIN*. És un sinònim de *D4*, nom del pin serigrafiat a la placa del D1 mini, que en realitat és el pin *GPIO2*. És a dir, si substituïm *LED_BUILTIN* al programa anterior per *D4* o *2*, el programa farà exactament el mateix.

Un error típic al programar el D1 mini és posar només el número de la pota serigrafiada. No és el mateix. Cal posar la D abans, ja que el número de pota serigrafiat no coincideix amb la pota física del xip:

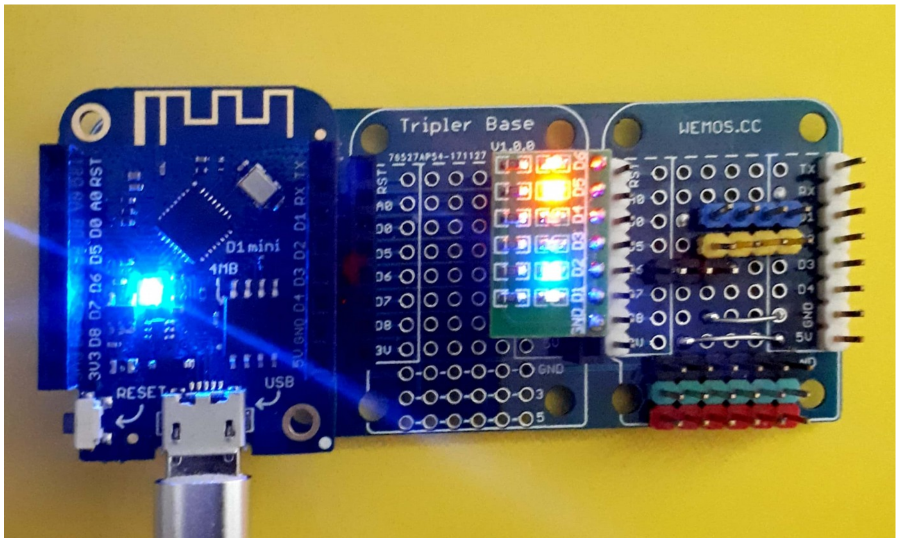


Ja podeu enviar el programa al D1 mini. El led blau integrat farà pampallugues.



Per altra banda, si us fixeu bé, aquest led reacciona a l'inrevés del que un espera. Quan posem un *LOW*, el led s'encén (durant 1s), i quan posem un *HIGH* s'apaga (durant 2s).

Aquest comportament només el trobem en aquest led blau, no afecta als leds connectats exteriorment. Ho podeu comprovar connectant un led extern entre la pota *D4* i *GND*.



Fixeu-vos com he col·locat el mòdul de leds, amb GND coincidint amb el GND de la base triple. Veureu com quan s'apaga el led integrat s'encén el led blanc, i a l'inrevés.

1-button shield

Aquest shield ens ofereix un polsador connectat a la pota *D3*. Quan premem el polsador, aquest connecta la entrada a *GND*. Es a dir, obtenim un *HIGH* si no prenem el polsador, i un *LOW* si ho fem.



Podeu comprovar el seu funcionament amb el meu exemple *button*:

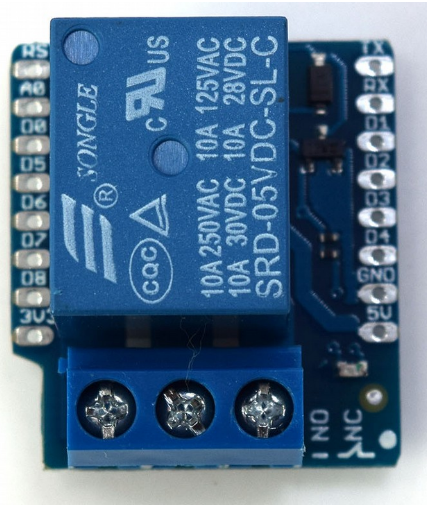
```
const int buttonPin = D3;
const int ledPin = BUILTIN_LED;

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  if (digitalRead(buttonPin) == HIGH) {
    digitalWrite(ledPin, HIGH);  // button released, LED off
  } else {
    digitalWrite(ledPin, LOW); // button pressed, LED on
  }
}
```



Relay shield V2.0.0



Aquest mòdul porta un relé (nosaltres el farem servir connectat a D7) amb el qual podem controlar fins i tot circuits a CA de 220V. Fixeu-vos que els límits de càrrega depenen de si aquesta la connectem a NO o a NC:

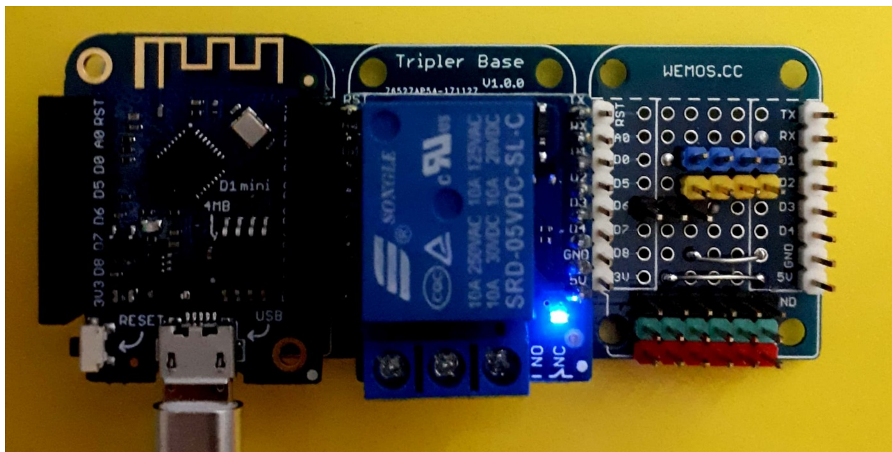
- NO: 5A(250VAC/30VDC), 10A(125VAC) MAX:1250VA/150W
- NC: 3A(250VAC/30VDC) MAX:750VA/90W

Podeu comprovar el seu funcionament amb el meu exemple *relay*. No oblideu canviar el *relayPin* a *D7*, tal com es mostra al llistat següent:

```
//const int relayPin = D1;  // comment for kit D1 mini R1
const int relayPin = D7;  // uncomment for kit D1 mini R1
const long interval = 2000;  // pause for two seconds

void setup() {
  pinMode(relayPin, OUTPUT);
}

void loop() {
  digitalWrite(relayPin, HIGH); // turn on relay
  delay(interval);              // pause
  digitalWrite(relayPin, LOW);  // turn off relay
  delay(interval);              // pause
}
```

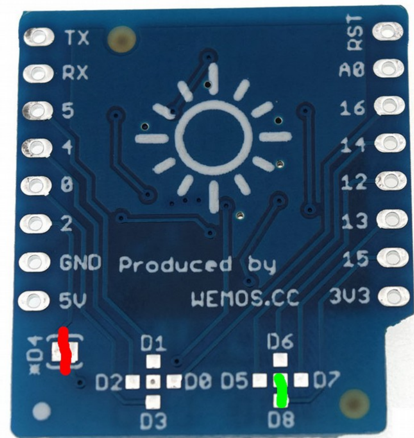


RGB shield

Aquest mòdul porta 7 leds RGB WS2812B, que fan servir un protocol d'un sol fil (per defecte connectat a D4). Nosaltres el fem servir connectat a D8.



La forma més senzilla d’utilitzar aquest shield és amb la llibreria Adafruit NeoPixel, que podem instal·lar des del gestor de llibreries de l’Arduino IDE.



Utilitzant la llibreria Adafruit NeoPixel, que podem instal·lar des del gestor de llibreries, podem provar aquest senzill exemple:

```
#include <Adafruit_NeoPixel.h>
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(7, D8, NEO_GRB + NEO_KHZ800);

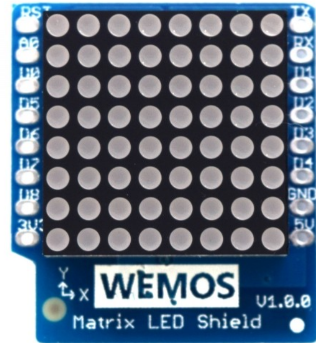
void setup() {
  pixels.begin(); // This initializes the NeoPixel library.
}

void loop() {
  for (int i=0; i<=255;i++){
        pixels.setPixelColor(0, pixels.Color(i,i,i )); // blanc.
        pixels.setPixelColor(1, pixels.Color(i,0,0 )); // vermell.
        pixels.setPixelColor(2, pixels.Color(i,i,0 )); // groc.
        pixels.setPixelColor(3, pixels.Color(0,i,0 )); // verd.
        pixels.setPixelColor(4, pixels.Color(0,i,i )); // cyan.
        pixels.setPixelColor(5, pixels.Color(0,0,i )); // blau.
        pixels.setPixelColor(6, pixels.Color(i,0,i )); // magenta.
        pixels.show(); // This sends the updated pixel color to the hardware.
        delay(10); // Delay for a period of time (in milliseconds).
  }
}
```



Matrix led shield

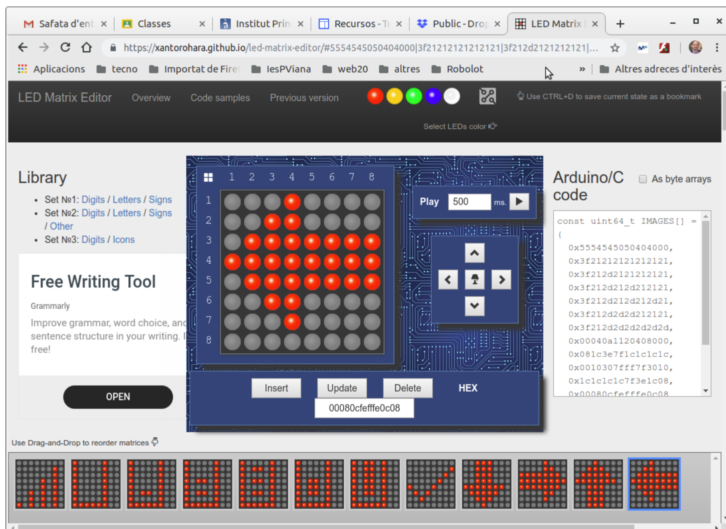
Matriu de 8x8 leds vermells amb 8 nivells d'intensitat. Utilitza D5 (*CLK*) i D7 (*DIN*)



Amb la llibreria *WEMOS Matrix LED Shield* és relativament senzill utilitzar aquest shield.

Amb la funció *dot(x,y,valor)* fixem els punts encesos (valor=1) i apagats (valor=0). Una vegada fixats, amb la funció display() s'actualitza la matriu de leds.

Gràcies a un editor online[6] podem fer animacions i definir pantalles:



6 Disponible a https://xantorohara.github.io/led-matrix-editor/

Al meu exemple *matrix_LED_animation* hem fet servir aquest editor, i us pot servir com a plantilla. Només cal canviar l'array *IMAGES[]:*

```
#include <WEMOS_Matrix_LED.h>
MLED mled(5); //set intensity=5

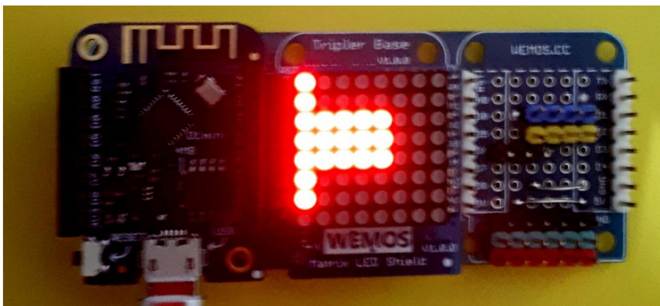
const uint64_t IMAGES[] = {
  0x00080cfefffe0c08,
  0x0004067f7f7f0604,
  0x0002033f3f3f0302,
  0x0001011f1f1f0101,
  0x0000000f0f0f0000,
  0x0000000707070000,
  0x0000000303030000,
  0x0000000101010000,
  0x0000000000000000
};
const int IMAGES_LEN = sizeof(IMAGES)/8;

void setup() {
}

void displayImage(uint64_t image) {
  for (int i = 0; i < 8; i++) {
    byte row = (image >> i * 8) & 0xFF;
    for (int j = 0; j < 8; j++) {
      mled.dot(j, 7-i, bitRead(row, j));
    }
  }
  mled.display();
}

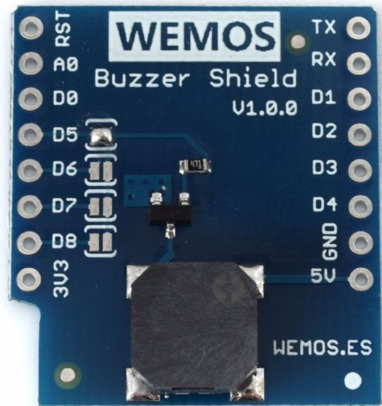
int i = 0;

void loop() {
  displayImage(IMAGES[i]);
  if (++i >= IMAGES_LEN ) {
    i = 0;
  }
  delay(100);
}
```



Buzzer shield

Afegeix un petit buzzer al nostre sistema.



Nosaltres treballem amb la connexió amb el pin *D5* (per defecte).

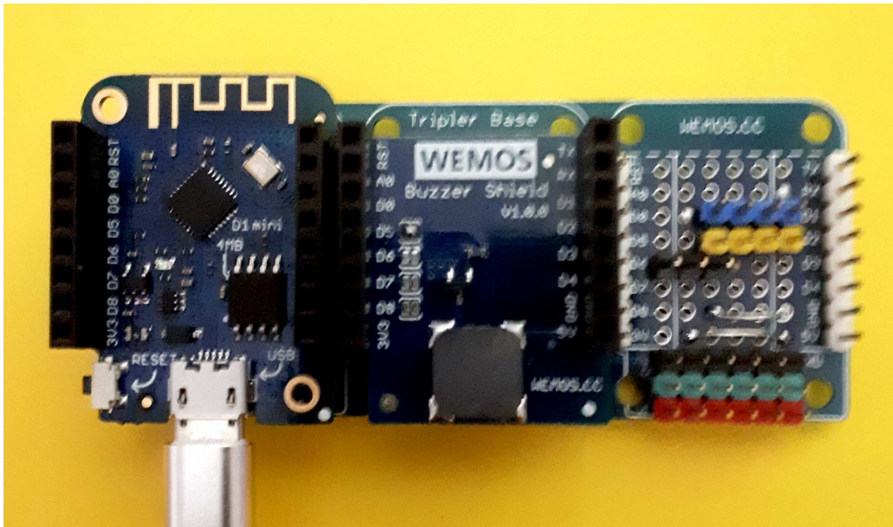
La implementació per al IDE Arduino del ESP8266 introdueix una funció molt útil per a aquest shield, *analogWriteFreq(freq)*, que permet fixar la freqüencia de la sortida i ens permet fer melodies.

Al meu exemple *sirena* podeu veure com utilitzar aquesta funció:

```
int buzzer=D5; //Buzzer control port, default D5

void setup() {
  pinMode(buzzer, OUTPUT);
  digitalWrite(buzzer, LOW);
}

void loop() {
  for (int i=0;i<5;i++){
      analogWriteFreq(988);        // 988 Hz, nota Do
      analogWrite(buzzer, 512);    // ona simètrica
      delay(700);                  // 0,7 s durada
      analogWrite(buzzer, 0);      // silenci
      delay(50);                   // 0,05 s
      analogWriteFreq(587);        // 587 Hz, nota Si
      analogWrite(buzzer, 512);
      delay(700);
      analogWrite(buzzer, 0);
      delay(50);
  }
  pinMode(buzzer, OUTPUT);
  digitalWrite(buzzer, LOW);      // silenci
  delay(5000);
}
```



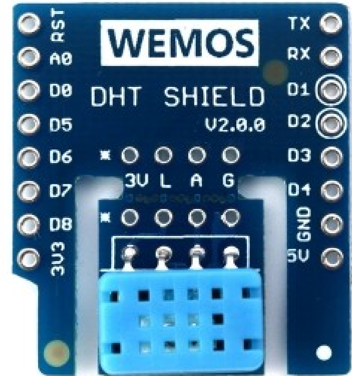
Podeu trobar informació molt útil a l’Annex 7: Taula de freqüències per a les notes musicals.

DHT shield

Sensor de temperatura i humitat DHT11 amb protocol I2C (potes *D1* i *D2*).

Temperatura: -20~60°C (±0.5°C)
Humitat: 20-95%RH (±5%RH)

És molt fàcil el seu ús amb la *WEMOS_DHT12_Arduino_Library*. Al meu exemple *DHT* es llegeix la temperatura i la humitat i es mostren pel canal sèrie




```
#include <WEMOS_DHT12.h>
DHT12 dht12;

void setup() {
  Serial.begin(115200);
}

void loop() {
  if(dht12.get()==0){
    Serial.print("Temperature in Celsius : ");
    Serial.println(dht12.cTemp);
    Serial.print("Temperature in Fahrenheit : ");
    Serial.println(dht12.fTemp);
    Serial.print("Relative Humidity : ");
    Serial.println(dht12.humidity);
    Serial.println();
  }
  delay(1000);
}
```

/dev/ttyUSB1 — Envia

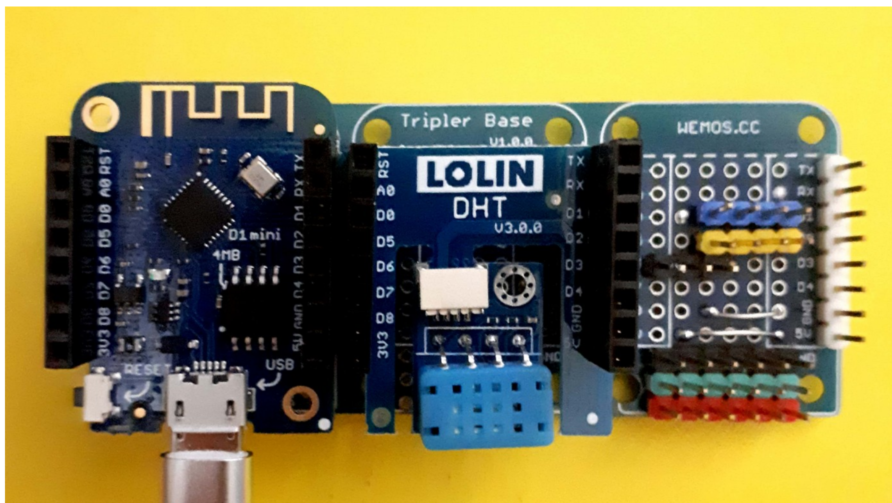
```
Temperature in Fahrenheit : 80.06
Relative Humidity : 33.20

Temperature in Celsius : 26.70
Temperature in Fahrenheit : 80.06
Relative Humidity : 33.20

Temperature in Celsius : 26.80
Temperature in Fahrenheit : 80.24
Relative Humidity : 33.20

Temperature in Celsius : 26.70
Temperature in Fahrenheit : 80.06
Relative Humidity : 33.10
```

☑ Desplaçament automàtic — Sense salts de línia — 115200 baud — Clear output

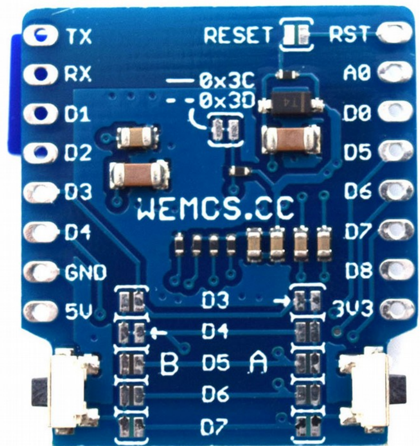


OLED Shield

Aquest mòdul porta un display gràfic OLED de 64x48 pixels mitjançant el protocol I2C (D1 *SCL*, D2 *SDA*).



Els nostres kits fan servir la versió 2.0.0, que inclou dos polsadors de connexió configurable (per defecte D3 i D4, que són els que fem servir).



Els millors resultats els he aconseguit amb la llibreria sparkfun/Micro_OLED_Breakout, encara que ha calgut un exemple modificat (*OLED*) configurat per a I2C i amb alguns canvis per a evitar problemes amb el WatchDog.

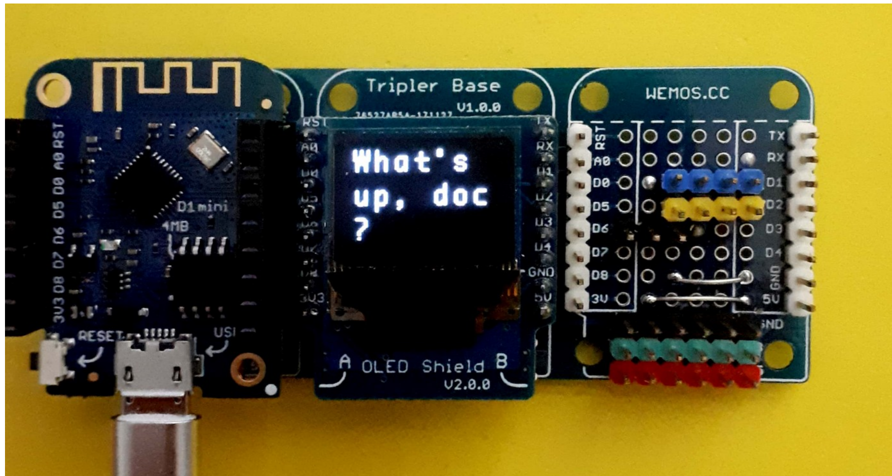
Tenim moltíssimes funcions, i inclouen funcions gràfiques molt avançades. Les més importants per mostrar un text a pantalla son *begin()*, *clear()*, *setFontType()*, *setCursor()*, *print()* i *println()*. Recordeu fer sempre un *display()* al final, o no es mostrarà res per pantalla.

El meu exemple *OLED* inclou tot tipus de funcions. Però per començar crec que serà més pràctic el meu exemple *OLED_test*:

```
#include <Wire.h>  // Include Wire if you're using I2C
#include <SFE_MicroOLED.h>  // Include the SFE_MicroOLED library
#define PIN_RESET 255  // Connect RST to pin 9
#define DC_JUMPER 0
MicroOLED oled(PIN_RESET, DC_JUMPER);    // I2C declaration

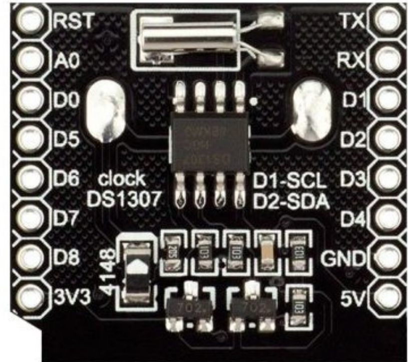
void setup()
{
  oled.begin();    // Initialize the OLED
  oled.clear(ALL); // Clear the display's internal memory
  oled.display();  // Display what's in the buffer (splashscreen)
  delay(1000);     // Delay 1000 ms
  oled.clear(PAGE); // Clear the buffer.
}

void loop()
{
  oled.clear(PAGE);
  oled.setFontType(1);
  oled.setCursor(0,0);
  oled.println("What's up, doc?");
  oled.display();
  delay(1500);
}
```



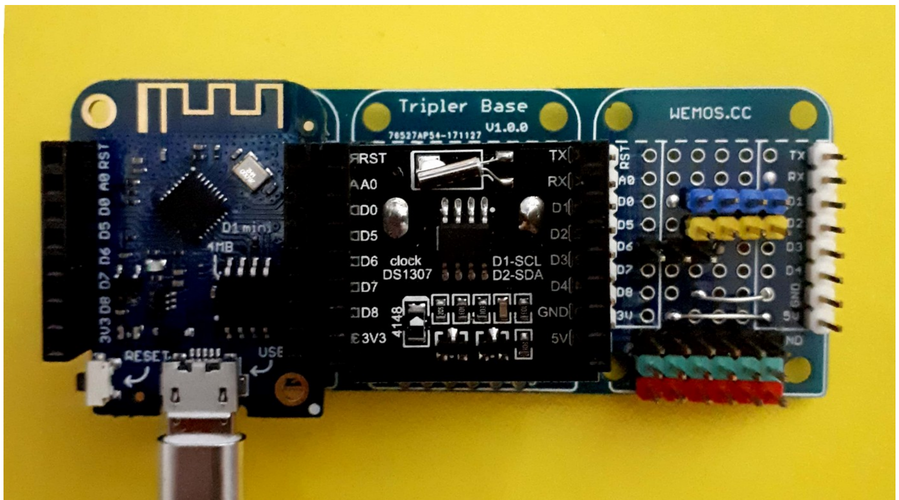
RTC shield

Aquest shield porta un rellotge en temps real (Real Time Clock) amb connexió I2C.



Aquest és un dels pocs shields que no fabrica Wemos / Lolin, si no RobotDyn, fabricant que també comercialitza una versió datalogger que, a més del rellotge I2C, inclou un lector microSD.

La llibreria *RTClib* (Adafruit) incorporada al Library manager funciona perfectament amb l'ESP8266, com podeu comprovar amb el meu exemple RTC_test.



```
#include <Wire.h>
#include "RTClib.h"
RTC_DS1307 RTC;

void setup () {
  Serial.begin(9600);
  Wire.begin();
  RTC.begin();
  // Check to see if the RTC is keeping time.  If it is, load the time from your computer.
  if (! RTC.isrunning()) {
    Serial.println("RTC is NOT running!");
    // This will reflect the time that your sketch was compiled
    RTC.adjust(DateTime(__DATE__, __TIME__));
  }
}
void loop () {
    DateTime now = RTC.now();
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print('/');
    Serial.print(now.year(), DEC);
    Serial.print(' ');
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();
    delay(1000);
}
```



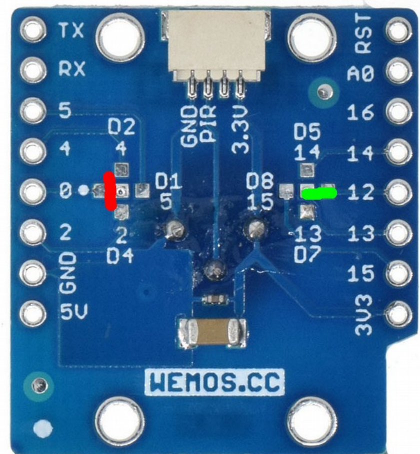
PIR shield

Aquest shield ens dona un detector de presència per calor (PIR). Quan detecta una persona, dona una sortida HIGH.



Nosaltres el tenim connectat a D6.

A tots els efectes el podem utilitzar com un interruptor d’entrada connectat a la pota D6, però heu de pensar que una vegada que es dispara, triga uns 3 s des de que no detecta gent a tornar la sortida a LOW.

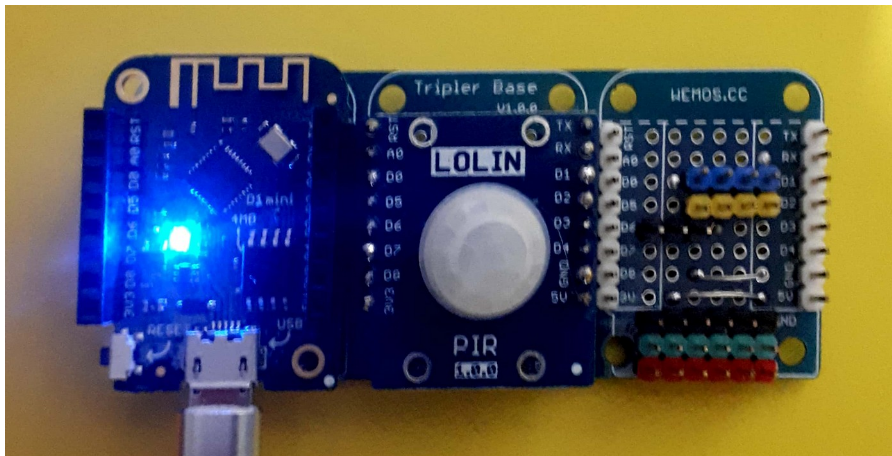


Podeu provar el meu exemple *PIR_test* per comprovar el seu funcionament.

```
const int buttonPin = D6;
const int ledPin = BUILTIN_LED;

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);  // LED off
}

void loop() {
  if (digitalRead(buttonPin) == HIGH) {
    digitalWrite(ledPin, LOW);  // people detected, LED on
  } else {
    digitalWrite(ledPin, HIGH); // LED off
  }
}
```

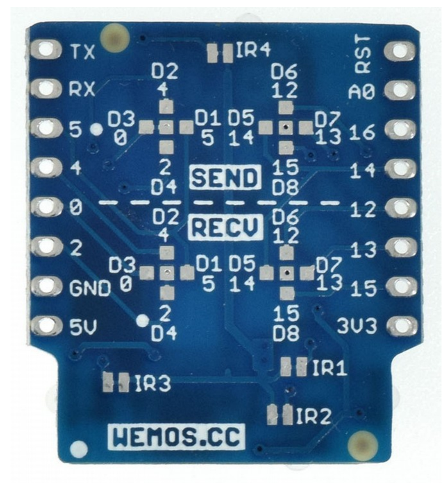


IR controller Shield

Aquest shield incorpora 4 emissors i 1 receptor de codis IR. Per defecte els emissors estan connectats a D3 i el receptor a D4. Nosaltres mantindrem aquestes connexions.



Amb la llibreria *IRremote ESP8266 Library* (es pot instal·lar des del gestor de llibreries) tenim diferents exemples per rebre i enviar codis IR.



Us deixo dos exemples:

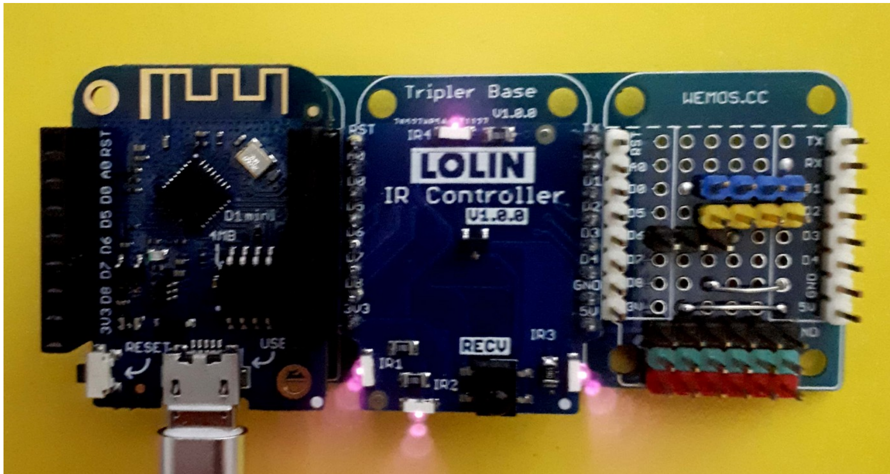
- *IRrx*, que rep i decodifica codis IR
- *IRtx*, que envia codis IR

```
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>

IRrecv irrecv(D4);
decode_results results;

void setup() {
  Serial.begin(115200);
  irrecv.enableIRIn();  // Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {
    // print() & println() can't handle printing long longs. (uint64_t)
    serialPrintUint64(results.value, HEX);
    Serial.println("");
    irrecv.resume();  // Receive the next value
  }
  delay(100);
}
```




```
#include <IRremoteESP8266.h>
#include <IRsend.h>
IRsend irsend(D3);

void setup() {
  irsend.begin();
}

void loop() {
  irsend.sendSony(0xa90, 12, 2);
  delay(2000);
}
```

Part II. Connectats amb xarxa

Fins ara no hem utilitzat la potència WiFi del nostre ESP8266. Ha arribat el moment de fer-la servir.

```
#include <ESP8266WiFi.h>
```

Aquesta senzilla línia dona sentit al títol d'aquest llibre. IoT. Internet of Things. Internet de les coses. I com podem dir IoT sense connexió a Internet? La veritat, n'estic fart de tallers, cursos, llibres i webs d'Arduino que s'anuncien com a IoT i la connexió a Internet no apareix enlloc.

Aquest, en canvi, és un llibre orientat a IoT. I amb aquest capítol veureu que va de debò!

Connexió com a estació

Per connectar-nos a una xarxa existent, com faríem amb qualsevol ordinador, hem de fer servir la funció

```
  WiFi.begin("network-name", "pass-to-network");
```

on cal canviar *network-name* pel nom de la nostre xarxa i *pass-to-network* per la contrasenya, que deixarem en blanc si es tracta d'una xarxa oberta.

Cal comprovar que hem aconseguit connectar-nos amb la funció WiFi.status(). Veiem un exemple:

```
#include <ESP8266WiFi.h>

void setup()
{
  Serial.begin(115200);
  Serial.println();

  WiFi.begin("network-name", "pass-to-network");

  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {}
```

Un servidor web. Controlant el relé via WiFi

La majoria de vegades el que volem es connecatr-nos al nostre ESP8266 i recollir dades o executar ordres. La millor forma és amb un servidor web. Per això caldrà afegir al principi

```
#include <ESP8266WebServer.h>
```

i definir un servidor al port 80:

```
ESP8266WebServer server(80);
```

Caldrà definir, típicament al *setup()* com reacciona el servidor a les diferents pàgines adjudicant a cada pàgina una funció amb

```
  server.on("PAGINA", FUNCIO);
```

i que fer en cas de no trobar la pàgina amb

```
  server.onNotFound(handleNotFound);
```

Una vegada definides les pàgines i la funció per quan es demani una no contemplada cal engegar el servidor amb

```
  server.begin();
```

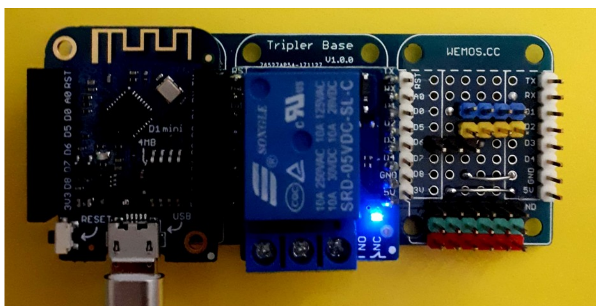
i estar pendent de les peticions entrants, típicament al loop() amb

```
  server.handleClient();
```

A les funcions definides podem generar una resposta amb

```
  server.send(codi-resposta, "text/plain", message);
```

Podem veure tot això funcionant al meu exemple *RelayRemote*:



```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = "la teva xarxa aqui";
const char* password = "contrasenya de xarxa aqui";

ESP8266WebServer server(80);
int estat;

void handleRoot() {
  server.send(200, "text/plain", "hello from esp8266!");
  }

void engegar() {
  digitalWrite(D1, true);
  estat=1;
  server.send(200, "text/plain", "ON");
  }

void apagar() {
  estat=0;
  server.send(200, "text/plain", "OFF");
  digitalWrite(D1, false);
  }

void canviar() {
  if(estat==0){
    digitalWrite(D1, true);
    estat=1;
    }
   else{
    digitalWrite(D1, false);
    estat=0;
    }
  server.send(200, "text/plain", "CHANGED");
  }

void handleNotFound(){
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET)?"GET":"POST";
  message += "\nArguments: ";message += server.args();
  message += "\n";
  for (uint8_t i=0; i<server.args(); i++){
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
    }
  server.send(404, "text/plain", message);
  }

void setup(void){
  pinMode(D1, OUTPUT);
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.println("");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
    }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  server.on("/", handleRoot);
  server.on("/on", engegar);
  server.on("/off", apagar);
  server.on("/change", canviar);
  server.on("/inline", [](){
    server.send(200, "text/plain", "this works as well");
    });
  server.onNotFound(handleNotFound);

  server.begin();
  Serial.println("HTTP server started");estat=0;
  digitalWrite(D1, false);
  }

void loop(void){
  server.handleClient();
  }
```

Connexió com a AP. Control WiFi d'un semàfor

El nostre ESP8266 pot crear una xarxa pròpia, a la que podem connectar-nos amb el nostre ordinador o mòbil.

En lloc del *WiFi.begin()* que fèiem servir la connexió com a estació, farem servir

```
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);
```

i ja el podrem fer servir.

Veiem com funciona amb el meu exemple *semafor_RGB_Wifi.ino*:




```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char *ssid = "Semaforo";
const char *password = "robotica";

ESP8266WebServer server ( 80 );

#include <Adafruit_NeoPixel.h>

#define PIN   D8
```

```
#define LED_NUM 7

Adafruit_NeoPixel leds = Adafruit_NeoPixel(LED_NUM, PIN, NEO_GRB + NEO_KHZ800);

int tr=2000;
int ty=500;
int tg=1500;

String mensaje = "";

//-----------CODIGO HTML PAGINA DE CONFIGURACION---------------
String pagina = "<!DOCTYPE html>"
"<html>"
"<head>"
"<title>Control de temps semafor</title>"
"<meta charset='UTF-8'>"
"<meta name='viewport' content='width=device-width'/>"
"</head>"
"<body>"
"<table style='width:100%'><form action='canviar' method='get'>"
"<tr><td>Temps vermell:</td><td><input type='range' name='fr' min='1' max='60'
value='3' step='1'></td></tr>"
"<tr><td>Temps groc:</td><td><input type='range' name='fy' min='1' max='60'
value='1' step='1'></td></tr>"
"<tr><td>Temps verd:</td><td><input type='range' name='fg' min='1' max='60'
value='2' step='1'></td></tr>"
"<tr><td></td><td><input type='submit' value='CANVIAR' /></td></tr>"
"</form></table>"
"</body>"
"</html>";

void espera(int temps) {
  unsigned long ara = millis();
  unsigned long seguent = ara + temps;
  delay(1); //refresh watchdog
  while (millis()<seguent){
      server.handleClient();
  }
}

void paginacanvi() {
  server.send(200, "text/html", pagina);
}

void canviar_temps() {
  tr=server.arg("fr").toInt()*1000;
  ty=server.arg("fy").toInt()*1000;
  tg=server.arg("fg").toInt()*1000;
  paginacanvi();
}


void led_set(uint8 R, uint8 G, uint8 B) {
  for (int i = 0; i < LED_NUM; i++) {
    leds.setPixelColor(i, leds.Color(R, G, B));
    leds.show();
    delay(50);
  }
}

void red_set() {
  leds.setPixelColor(1, leds.Color(100, 0, 0));
  leds.setPixelColor(0, leds.Color(10, 10, 0));
  leds.setPixelColor(4, leds.Color(0, 10, 0));
```

```
  leds.show();
  espera(tr);
}

void yellow_set() {
  leds.setPixelColor(1, leds.Color(10, 0, 0));
  leds.setPixelColor(0, leds.Color(50, 50, 0));
  leds.setPixelColor(4, leds.Color(0, 10, 0));
  leds.show();
  espera(ty);
}

void green_set() {
  leds.setPixelColor(1, leds.Color(10, 0, 0));
  leds.setPixelColor(0, leds.Color(10, 10, 0));
  leds.setPixelColor(4, leds.Color(0, 100, 0));
  leds.show();
  espera(tg);
}


void setup() {
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);
  server.on("/", paginacanvi);
  server.on("/canviar", canviar_temps);
  server.begin();
  leds.begin();
  led_set(0, 0, 0);
}


void loop() {

  green_set();
  yellow_set();
  red_set();

}
```

Fixeu-vos que al programa en lloc de *delay()* faig servir *espera()*, una funció que he definit i que mentre espera el temps marcat en ms està pendent de les peticions web amb *handleClient()*.

AP + estació

És possible generar una xarxa com a AP i, a la vegada, estar connectat a una altra xarxa com a estació:

```
WiFi.mode(WIFI_AP_STA);
WiFi.softAP(ssid1, password1);
WiFi.begin(ssid2, password2);
```

Encara que són dues xarxes diferents, farà servir un únic canal: el que utilitza la xarxa a la que es connecta com a estació.

Una aplicació interessant d'aquesta doble connexió és la creació de xarxes de malla. Saida Sillo va fer un TR que vaig tutoritzar amb el títol «Xarxa detectora d'incendis forestals»[7] on utilitzava aquestes xarxes per cobrir tot un bosc amb sensors i permetre enviar una alerta a internet. Cada sensor és, a la vegada, AP i estació dels seus veïns. Estació per transmetre l'alerta, AP per rebre les alertes dels seus veïns i passar la informació a la resta com a estació.

Una altra aplicació freqüent és utilitzar l'accés AP com a porta del darrera per a configurar el dispositiu quan perd la connexió a la xarxa com a estació. D'aquesta forma podem canviar el nom de la xarxa i contrasenya amb que es connecta.

També és freqüent aquesta doble connexió en un dispositiu mòbil (al que accedirem com a AP des del mòbil o tablet fora de casa) al que volem accedir des d'un ordinador fix a casa (és farragós haver de canviar la configuració d'una torre per accedir al nostre dispositiu, impossible si la torre es connecta per cable a la xarxa). Un exemple seria una càmera digital o un enregistrador de dades, controlable des del mòbil i amb unes fotografies i dades que volem descarregar després a l'ordinador de casa.

7 Disponible a https://github.com/jorts64/kit-D1-mini/blob/master/projectes/2017-18%20TR/xarxadetectora.pdf

Estació o AP?

És interessant no canviar el *WiFi.mode()* si és possible. El nom de xarxa i la contrasenya s'emmagatzemen a la memòria *RTC* del ESP8266, de forma que no es perden amb un *RST* o apagat del sistema[EAC01]. Sempre i quan no canviem el *WiFI.mode()*. Aquesta estratègia evita haver de guardar nosaltres aquestes dades.

Utilitzar *WIFI_AP* és interessant en entorns hostils: tallers, conferències, demos, on de vegades l'accés a la xarxa és complicat. Penseu que els tipus de criptografia implementats són limitats. O les xarxes tenen limitada l'accés a IPs o ports no autoritzats.

Però amb *WIFI_AP* no tindrem accés a internet, limitant l'ús de llibreries javascript ubicades al núvol o l'enviament de dades a servidors. En aquest cas ens interessa *WIFI_STA*, encara que sigui generant una xarxa amb el nostre mòbil. Per cert, si la xarxa a la que ens connectem no té contrasenya hem de posar

```
WiFi.begin(ssid, "");
```

Finalment, recordeu que també podem apagar la WiFI (i estalviar bateria) amb

```
  WiFi.mode(WIFI_OFF);
```

Al meu projecte «Càmera tèrmica»[8] teniu un exemple de com engegar amb botons la WiFi, tant com *WIFI_AP* com a *WIFI_STA*. Fixeu-vos en la funció *initWifi()* al meu codi.

8 Disponible a https://github.com/jorts64/kit-D1-mini/tree/master/projectes/jorts/camera%20termica

ThingSpeak: enregistrar les nostres dades al núvol

ThingSpeak és un servei gratuït que ens permet publicar al núvol les nostres dades i visualitzar-les de forma gràfica. A més a més permet fer l'anàlisi amb la potència de MATLAB.

Primer de tot crearem un compte a http://thingspeak.com i un canal on centralitzar les nostres mesures. Prenem nota del canal i del *APIKEY* associat per escriure'n dades.

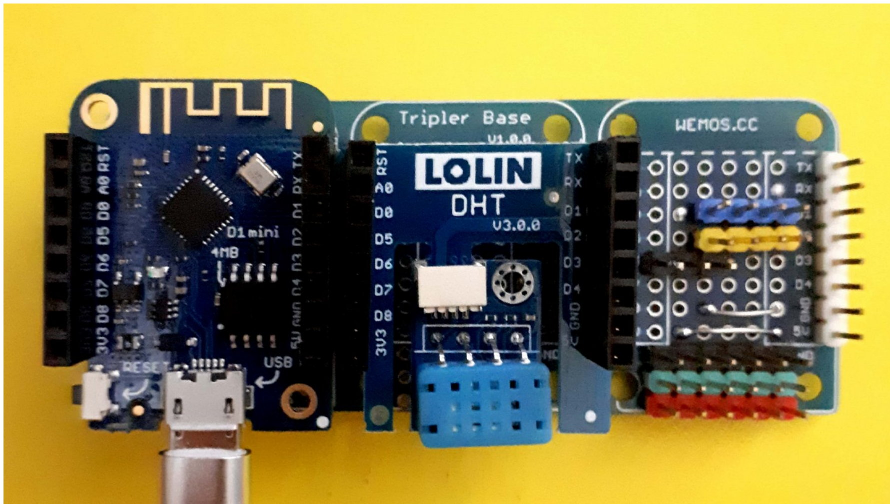
Haurem d'instal·lar la llibreria *thinkspeak a l'Arduino IDE* (https://github.com/mathworks/thingspeak-arduino). Ho podeu fer amb el gestor de llibreries de l'IDE.

Podeu provar-lo amb el meu exemple *thingspeak*:

```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"
#include <WEMOS_DHT12.h>
DHT12 dht12;
const char *ssid = "Nom de la xarxa WiFi aqui";
const char *password = "Contrasenya de la WiFi aqui";
WiFiClient  client;
unsigned long myChannelNumber = 135405; //canvia pel teu canal
const char * myWriteAPIKey = "API key aqui";
const int led = BUILTIN_LED;  // internal blue led

void setup () {
  pinMode ( led, OUTPUT );
  digitalWrite ( led, HIGH );
  WiFi.begin ( ssid, password );
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
  }
  ThingSpeak.begin(client);
}

void loop () {
  digitalWrite ( led, LOW );
  dht12.get();
  ThingSpeak.writeField(myChannelNumber, 1, dht12.cTemp, myWriteAPIKey);
  digitalWrite ( led, HIGH );
  delay(60000); // ThingSpeak will only accept updates every 15 seconds.
}
```



Quant tenim més d'una dada el mecanisme és lleugerament diferent. Primer preparem les dades i després les enviem de cop. Veieu el meu exemple thinspeak2c:

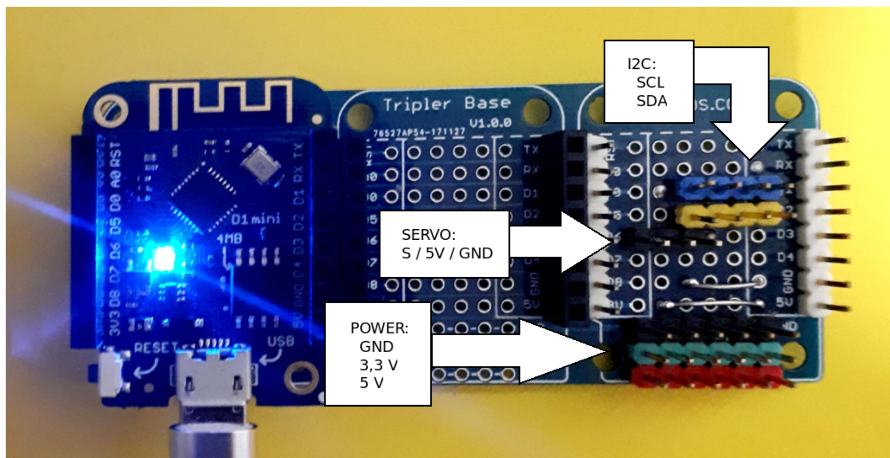
```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"
#include <WEMOS_DHT12.h>
DHT12 dht12;
const char *ssid = "Nom de la xarxa WiFi aqui";
const char *password = "Contrasenya de la WiFi aqui";
WiFiClient  client;
unsigned long myChannelNumber = 185812; //canvia pel teu canal
const char * myWriteAPIKey = "API key aqui";
const int led = BUILTIN_LED;  // internal blue led
int t;

void setup () {
  pinMode ( led, OUTPUT );
  digitalWrite ( led, HIGH );
  WiFi.begin ( ssid, password );
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
  }
  ThingSpeak.begin(client);
}

void loop () {
  digitalWrite ( led, LOW );
  dht12.get();
  ThingSpeak.setField(1,dht12.cTemp);
  ThingSpeak.setField(2,dht12.humidity);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  digitalWrite ( led, HIGH );
  delay(60000); // ThingSpeak will only accept updates every 15 seconds.
}
```

Part III. Utilitzant components externs

Hem sacrificat una de les 3 columnes de la triple base per posar-hi connectors mascle. Ara veurem la seva utilitat.



Les sortides no haurien de donar cap problema, però les entrades poden ser molt complicades si oblidem els pull-ups i pull-downs que porta la placa integrats, o el divisor de tensió integrat a l'entrada analògica i que pel meu gust es de massa baixa impedància.

Recordeu que quan treballem amb components I2C cal que siguin alimentats a 3,3V. Si necessitem treballar amb un component I2C a 5V, ens caldrà un adaptador de nivells pels senyals SCL i SDA. Connectar un component I2C de 5V sense aquest adaptador no només fa que la senyal no sigui fiable (el senyal de 3,3V de l'ESP8266 quedaria fora de les especificacions del bus I2C a 5V), a més a més podem fer malbé els components I2C que funcionen a 3,3V.

També recordeu que al bus I2C calen pull-ups a 3,3V a SCL i SDA. La majoria de shields i mòduls I2C porten aquests pull-ups integrats.

Per tant quan tenim un mòdul I2C farem les següent connexions:

GND → GND	(connector negre de 6 pins)
VCC → 3,3V	(connector verd de 6 pins)
SCL → D1	(connecor blau de 4 pins)
SDA → D2	(connector groc de 4 pins)

Si treballem amb un component I2C i no estem segurs de la seva adreça, podem utilitzar el programa *I2Cscanner*, que en farà un llistat dels dispositius I2C trobats:

```
#include <Wire.h>
void setup() {
  Wire.begin();
  Serial.begin(9600);
  while (!Serial);             // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}

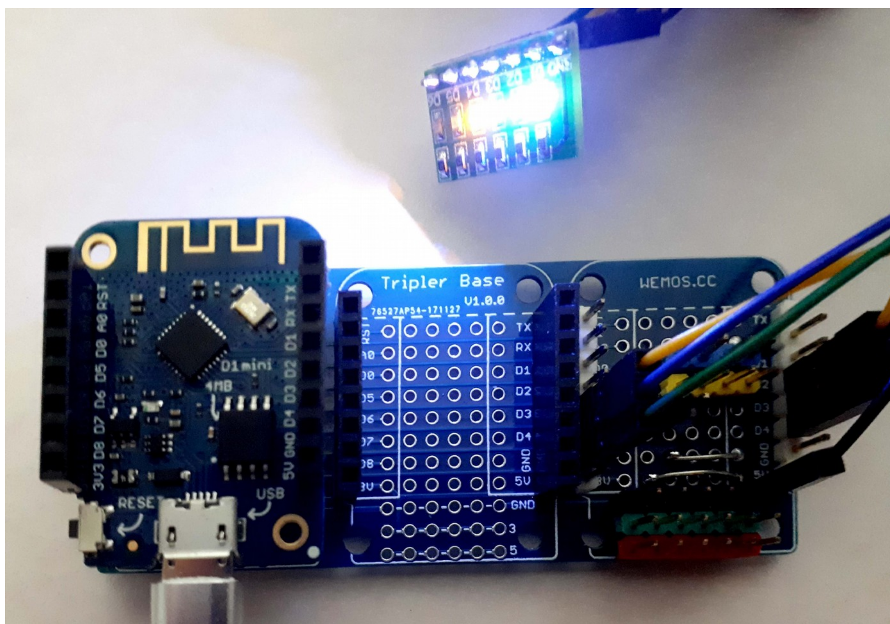
void loop() {
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");
      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");
  delay(5000);           // wait 5 seconds for next scan
}
```

Mòdul 6 LEDs

Ers tracta d'un mòdul molt econòmic i compacte, de 6 leds de diversos colors, cadascú amb la seva resistència de 1 kΩ i que comparteixen negatiu.

Tots els pins D0-D8 funcionen perfectament com a sortida. Això si, donen 3,3 V en estat *HIGH*. Recordeu que l'ESP8266 funciona a aquest voltatge.

Utilitzarem els cables Dupont femella-femella per connectar el mòdul.



Veiem un programa senzill:

```
oid setup() {
  pinMode(D2,OUTPUT);
  pinMode(D3,OUTPUT);
  pinMode(D4,OUTPUT);
  pinMode(D6,OUTPUT);
  pinMode(D7,OUTPUT);
  pinMode(D8,OUTPUT);
}

void loop() {
  digitalWrite(D2,LOW);
  digitalWrite(D3,LOW);
  digitalWrite(D4,LOW);
  digitalWrite(D6,LOW);
  digitalWrite(D7,LOW);
  digitalWrite(D8,LOW);
  delay(1000);
  digitalWrite(D2,HIGH);
  delay(1000);
  digitalWrite(D3,HIGH);
  delay(1000);
  digitalWrite(D4,HIGH);
  delay(1000);
  digitalWrite(D6,HIGH);
  delay(1000);
  digitalWrite(D7,HIGH);
  delay(1000);
  digitalWrite(D8,HIGH);
  delay(1000);
}
```

No us ha d’estranyar que quan el led connectat a D4 estigui apagat s’encengui el led blau integrat a l’ESP8266. Recordeu que aquest led està polaritzat a l’inrevés.

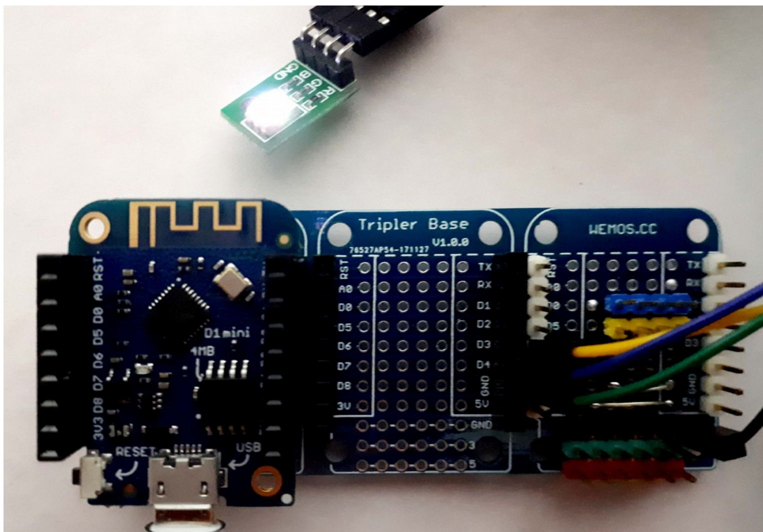
Mòdul LED RGB

Molt semblant al mòdul de 6 leds, cada entrada d'aquest led porta la seva resistència limitadora (270 Ω per verd i blau, 470 Ω pel vermell) i comparteixen negatiu.

Tots els pins D1-D8 permeten PWM. Ho comprovem amb un senzill programa:

```
void setup() {
  pinMode(D6,OUTPUT);
  pinMode(D7,OUTPUT);
  pinMode(D8,OUTPUT);
}

void loop() {
  int i,j,k;
  for (i=0;i<256;i+=10) {
    analogWrite(D6,i);
    for (j=0;j<256;j+=10){
      analogWrite(D7,j);
      for (k=0;k<256;k+=10) {
        analogWrite(D8,k);
        delay(1);
      }
    }
  }
}
```

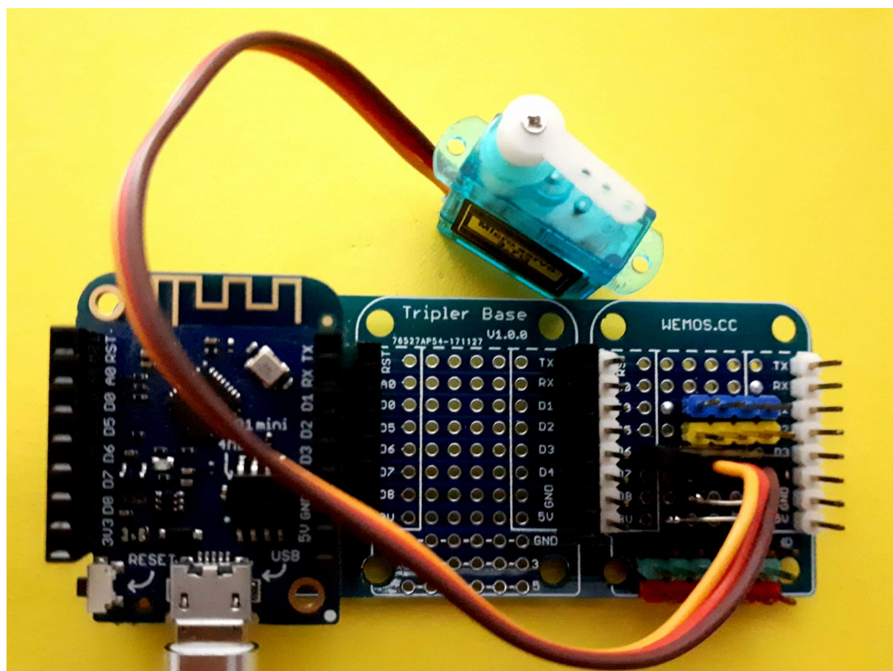


Microservo 3,7g

Aquest servo, econòmic (no tant com el de 9g, però ocupa menys a la capsa del kit) ens mostra com utilitzar el connector estàndard de servo (S / 5V / GND) que hem creat a la triple base.

Un dels avantatges dels servos és que agafen l'alimentació de forma independent. Per això el connector que hem fet té al mig 5V, encara que el microcontrolador ESP8266 funcioni a 3,3V.

Fixeu-vos que el cable taronja (senyal de control) ha de quedar a l'esquerra.



Podem provar-lo amb l'exemple de la llibreria servo específica pel ES8266. Caldrà canviar el pin per D6.

```
/* Sweep
  by BARRAGAN <http://barraganstudio.com>
  This example code is in the public domain.

  modified 28 May 2015
  by Michael C. Miller
  modified 8 Nov 2013
  by Scott Fitzgerald

  http://arduino.cc/en/Tutorial/Sweep
*/

#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards


void setup() {
  myservo.attach(D6);  // attaches the servo on GIO2 to the servo object
}

void loop() {
  int pos;

  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);     // tell servo to go to position in variable 'pos'
    delay(15);              // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);     // tell servo to go to position in variable 'pos'
    delay(15);              // waits 15ms for the servo to reach the position
  }
}
```

Mòdul 4 polsadors

Mòdul compacte i econòmic, porta quatre polsadors que connecten a negatiu.

Com D8 porta un pull-up a negatiu, millor no utilitzar aquest pin.

D3 i D4 porten pull-ups a positiu. En principi els podem fer servir amb

```
pinMode(D3, INPUT);
pinMode(D4, INPUT);
```

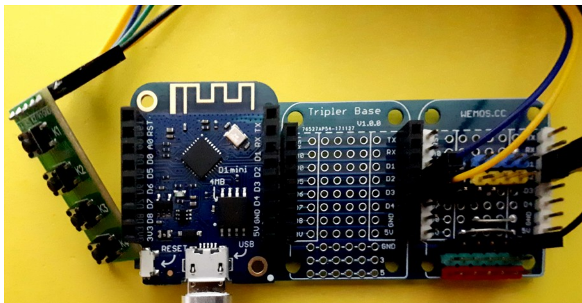
però no es recomanable fer servir aquestes entrades ja que si es fa un RST amb l'entrada a LOW no s'inicia el programa.

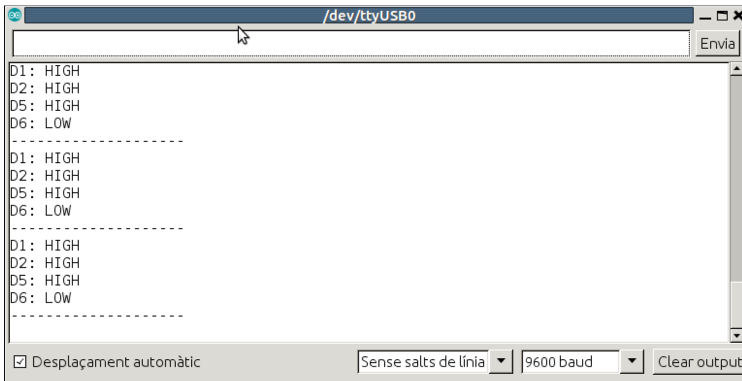
D0 no porta pull-up intern configurable, per tant tampoc el farem servir.

Finalment, D1, D2, D5, D6 i D7 permeten configurar un pull-up intern. Són les entrades que farem servir amb aquest mòdul amb les ordres

```
pinMode(D1, INPUT_PULLUP);
pinMode(D2, INPUT_PULLUP);
pinMode(D5, INPUT_PULLUP);
pinMode(D6, INPUT_PULLUP);
pinMode(D7, INPUT_PULLUP);
```

Veiem un petit exemple que ens mostra l'estat de les entrades pel canal sèrie:






```
void setup() {
  pinMode(D1, INPUT_PULLUP);
  pinMode(D2, INPUT_PULLUP);
  pinMode(D5, INPUT_PULLUP);
  pinMode(D6, INPUT_PULLUP);
  Serial.begin(9600);
}

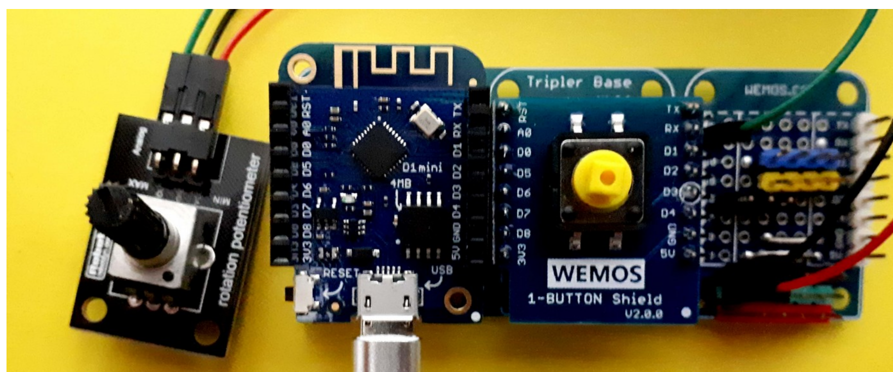
void loop() {
  Serial.print("D1: ");
  if (digitalRead(D1)) {
    Serial.println("HIGH");
  }
  else {
    Serial.println("LOW");
  }
  Serial.print("D2: ");
  if (digitalRead(D2)) {
    Serial.println("HIGH");
  }
  else {
    Serial.println("LOW");
  }
  Serial.print("D5: ");
  if (digitalRead(D5)) {
    Serial.println("HIGH");
  }
  else {
    Serial.println("LOW");
  }
  Serial.print("D6: ");
  if (digitalRead(D6)) {
    Serial.println("HIGH");
  }
  else {
    Serial.println("LOW");
  }
  Serial.println("--------------------");
  delay(2000);
 }
```

Mòdul potenciòmetre

Aquest mòdul porta un potenciòmetre de 10 k Ω que podem utilitzar com a divisor de tensió i llegir el seu valor amb l'entrada analògica A0.

Les entrades analògiques són un problema en l'ESP8266. Només en tenim una, i internament treballa en el rang 0V:1V. A la placa D1 mini porta un divisor de tensió integrat per agafar 1/3 com a mostra de la tensió d'entrada, amb la qual cosa el rang passa a ser de 0V : 3,3V. El problema és que el divisor de tensió està fet amb resistències de valor molt petit pel meu gust: 100 kΩ i 220 kΩ, deixant una impedància d'entrada baixa. Si bé podem llegir valors de consigna amb aquest potenciòmetre, el fa inservible per a altres aplicacions, com ara la utilització de leds com a sensors de llum.

Amb el meu exemple *Analog2WiFi* podem llegir el valor del potenciòmetre i comparar-lo amb un valor de referència, deforma que en superar-lo encengui el led integrat. A més a més, si activem el polsador del shield 1-button genera una xarxa WiFi i mostra una senzilla gràfica de 400 lectures amb codi SVG[9]:



9 Podeu conèixer més sobre el codi SVG (Scalable Vector Graphics) a HTML5 a https://www.w3schools.com/html/html5_svg.asp

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
const char *ssid = "ESPap";
const char *password = "robotica";
ESP8266WebServer server(80);

void handleRoot() {
 server.send(200, "text/html", "<html><head></head><body><a 
href='graf.svg'>Grafica</a><br><a href='rst'>Reset</a></body></html>");
}

const int buttonPin = D3;
const int ledPin = BUILTIN_LED;

void setup() {
  delay(1000);
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
}

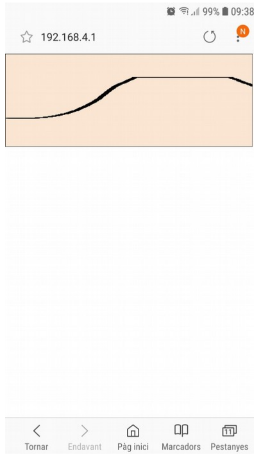
void loop() {
  if (digitalRead(buttonPin)==LOW) {
    prova();
  }
  else {
    if (analogRead(A0)>512) {
        digitalWrite(ledPin, LOW);
        delay(100);
        digitalWrite(ledPin, HIGH);
        delay(100);
    }
  }
  delay(1);
}

void prova() {
  digitalWrite(ledPin, LOW);
  WiFi.softAP(ssid, password);
  server.on("/", handleRoot);
  server.on("/graf.svg", grafica);
  server.on("/rst", reinicia);
  server.begin();
  while (true) {
    server.handleClient();
  }
}

void grafica() {
  String out = "";
  char temp[1000];
  out += "<svg xmlns=\"http://www.w3.org/2000/svg\" version=\"1.1\" 
width=\"400\" height=\"150\">\n";
  out += "<rect width=\"400\" height=\"150\" fill=\"rgb(250, 230, 210)\" 
stroke-width=\"1\" stroke=\"rgb(0, 0, 0)\" />\n";
  out += "<g stroke=\"black\">\n";
  int y = analogRead(A0)/10;
  delay(5);
  for (int x = 1; x < 399; x++) {
    int y2 = analogRead(A0)/10;
    delay(5);
    sprintf(temp, "<line x1=\"%d\" y1=\"%d\" x2=\"%d\" y2=\"%d\" stroke-
width=\"1\" />\n", x, 140 - y, x + 10, 140 - y2);
```

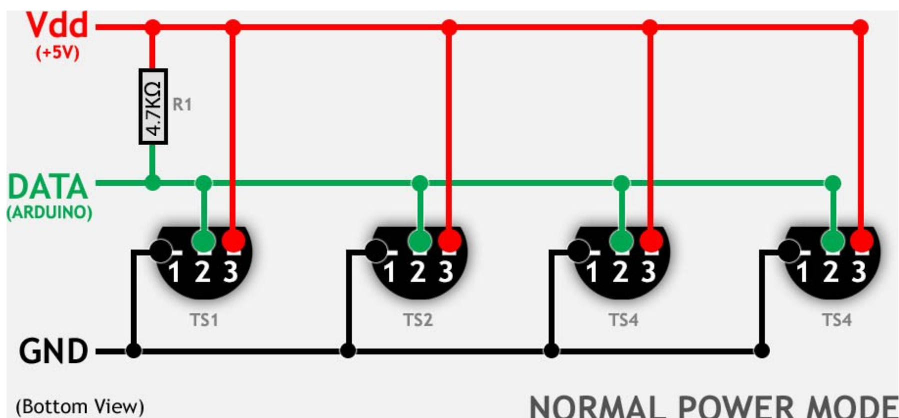
```
    out += temp;
    y = y2;
  }
  out += "</g>\n</svg>\n";
  server.send ( 200, "image/svg+xml", out);
}

void reinicia() {
  ESP.restart();
}
```



Mòdul sensor temperatura DS18B20

Aquest sensor és molt interessant. De fet, un dels meus favorits. Dona la temperatura amb una precisió (± 0,5 °C) i resolució (de 9 a 12 bits) altíssimes en un rang molt ampli de temperatures (de -55 °C a +85 °C). Essent digital fa servir només 3 fils (Vcc, GND i senyal) amb un bus propi on podem connectar diversos sensors. Com cada sensor porta gravat amb làser un codi únic de 64 bits a la seva ROM en teoria podríem connectar un nombre il·limitat de sensors al mateix pin.



Tots els pins D1-D8 tenen suport per a aquest bus OneWire, però com el pin D8 porta un pull-down no el podem fer servir per a aquest tipus de sensor. Per altre banda, el led integrat connectat a D4 ens pot donar problemes, així que millor no connectar-hi un DS18B20.

Existeix un shield pel D1 mini amb aquest sensor (veure pàg. 127). Malauradament va connectat al pin D2, amb la qual cosa col·lisiona amb el bus I2C, i no porta un connector per afegir altres sensors externs. Per això el kit porta un mòdul extern, que podem connectar a qualsevol dels pins D1, D2, D3, D5, D6 o D7 que tinguem lliure i

permet soldar un segon connector de 3 pins per connectar més sensors al mateix bus.

Per utilitzar aquest sensor cal tenir les llibreries *OneWire* i *DallasTemperature*, que podeu afegir amb el gestor de llibreries.

A l’exemple DS18B20_AJAX utilitzem aquest sensor per mostrar via web la temperatura cada segon:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <OneWire.h>
#include <DallasTemperature.h>

const char *ssid = "nom de la xarxa";
const char *password = "contrasenya";

ESP8266WebServer server ( 80 );
#define ONE_WIRE_BUS D7  // DS18B20 pin
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature DS18B20(&oneWire);
float t;

const char *pagina = "<html>\
<head>\
  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>\
  <meta name='viewport' content='width=device-width'> \
  <title>AJAX Test</title>\
  <style>body{padding:0;margin:0;background:#999}</style>\
  <script>\
function loadTemp() {\
  var xhttp = new XMLHttpRequest();\
  var temp = 0;\
  xhttp.open('GET', 'ajax_info.txt', false);\
  xhttp.send();\
  temp = parseFloat(xhttp.responseText);\
  return (temp);\
} \
  </script>\
</head>\
<body>\
<p>Temperatura:</p>\
<p id='demo'></p>\
<script>\
var myVar = setInterval(myTimer, 1000);\
function myTimer() {\
  document.getElementById('demo').innerHTML = loadTemp();\
}\
</script>\
</body>\
</html>";

void handleRoot() {
  server.send ( 200, "text/html", pagina );
}

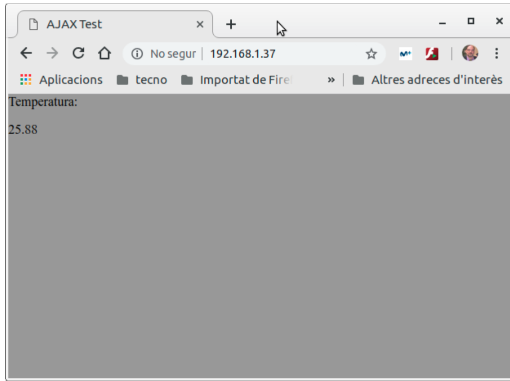
void handleNotFound() {
```

```
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += ( server.method() == HTTP_GET ) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for ( uint8_t i = 0; i < server.args(); i++ ) {
    message += " " + server.argName ( i ) + ": " + server.arg ( i ) + "\n";
  }
  server.send ( 404, "text/plain", message );
}

void handleAjax(){
  DS18B20.requestTemperatures();
  t = DS18B20.getTempCByIndex(0);
  Serial.print("Temperature: ");
  Serial.println(t);
  String out = String(t);
  server.send ( 200, "text/txt", out);
}

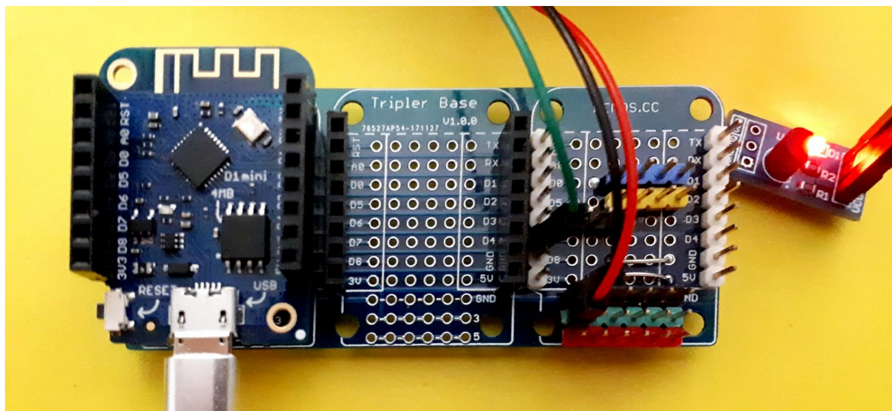
void setup ( void ) {
  Serial.begin ( 115200 );
  WiFi.begin ( ssid, password );
  DS18B20.begin();
  Serial.println ( "" );
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
    Serial.print ( "." );
  }
  Serial.println ( "" );
  Serial.print ( "Connected to " );
  Serial.println ( ssid );
  Serial.print ( "IP address: " );
  Serial.println ( WiFi.localIP() );
  server.on ( "/", handleRoot );
  server.on ("/ajax_info.txt", handleAjax );
  server.onNotFound ( handleNotFound );
  server.begin();
  Serial.println ( "HTTP server started" );
}

void loop ( void ) {
  server.handleClient();
}
```



Fixeu-vos com amb AJAX[10] podem actualitzar una pàgina web minimitzant el trànsit de dades. A la part HTML5 bàsicament ens cal una funció javascript per agafar les dades (*loadTemp()* en el nostre cas) i trucar-la periòdicament[11] per actualitzar la pàgina (funció myTimer() activada per *setInterval(myTimer, temps en ms)*).

Al codi Arduino ens cal una funció (*handleAjax()* en el nostre cas) que llegeixi la temperatura i la lliuri en el format adequat, activada quan demanem una pàgina en concret (*ajax_info.txt* en el nostre cas).



10 Podeu trobar més informació sobre com utilitzar AJAX al vostre codi javascript a https://www.w3schools.com/js/js_ajax_intro.asp

11 Podeu trobar més informació sobre *setInterval()* i els seus paràmetres a https://www.w3schools.com/jsref/met_win_setinterval.asp

Mòdul sensor de llum I2C BH1750FVI

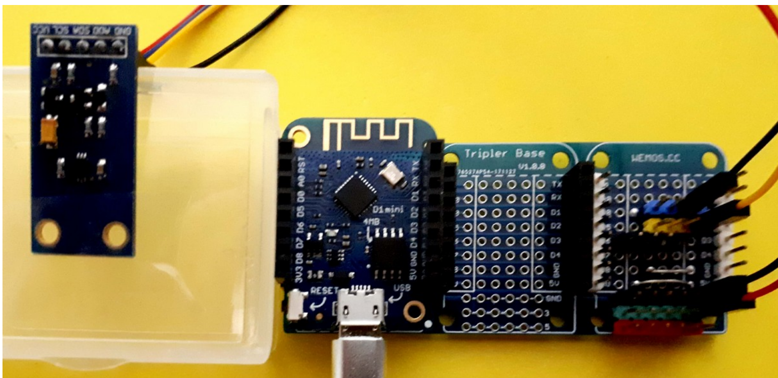
Aquest sensor pot treballar amb 3,3V. També està disponible com a shield del D1 mini.

El principal avantatge d'aquest sensor és que ens dona la llum directament en lux. La precisió (que pot arribar a ser de ± 0,5 lx) depèn del mode d'operació, i repercuteix en el temps de mesura. L'exemple BH1750 utilitza el mode d'alta resolució continu, amb un temps de mesura de 120 ms i una precisió de ± 1 lx:

```
//Install [claws/BH1750 Library](https://github.com/claws/BH1750) first.
#include <Wire.h>
#include <BH1750.h>
BH1750 lightMeter(0x23);

void setup(){
  Serial.begin(9600);
  Wire.begin();
  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE)) {
    Serial.println(F("BH1750 Advanced begin"));
  }
  else {
    Serial.println(F("Error initialising BH1750"));
  }
}

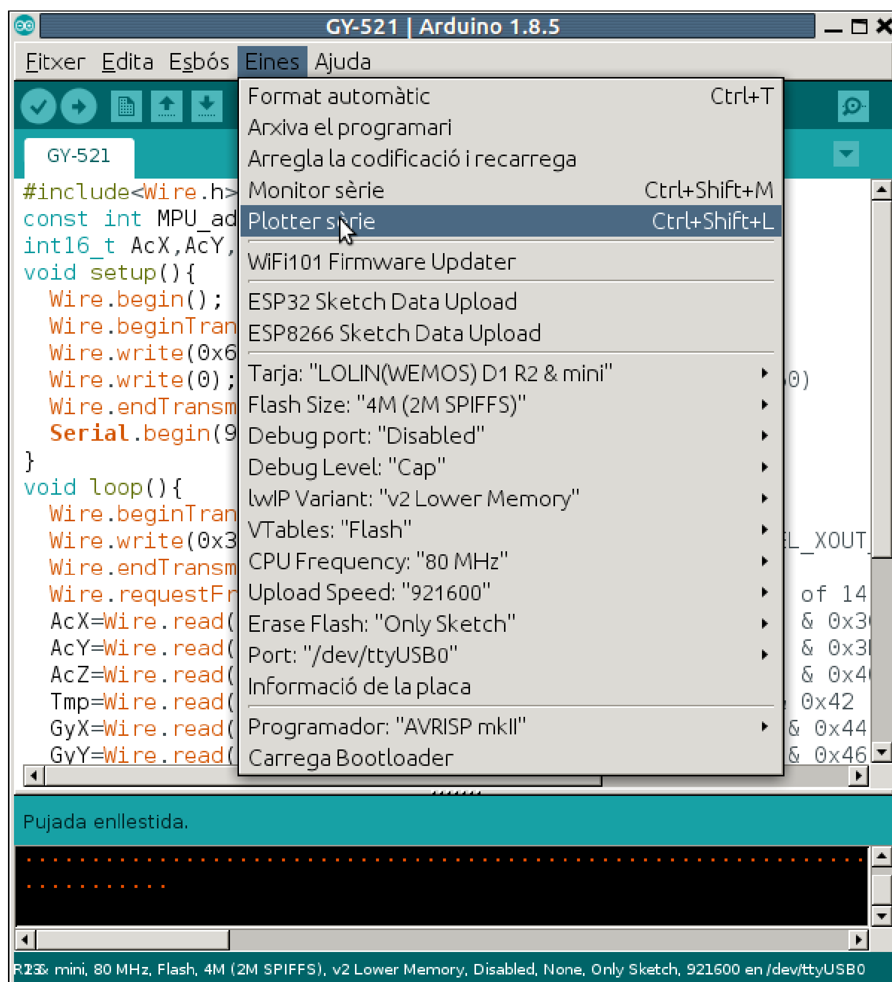
void loop() {
  uint16_t lux = lightMeter.readLightLevel();
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(1000);
}
```

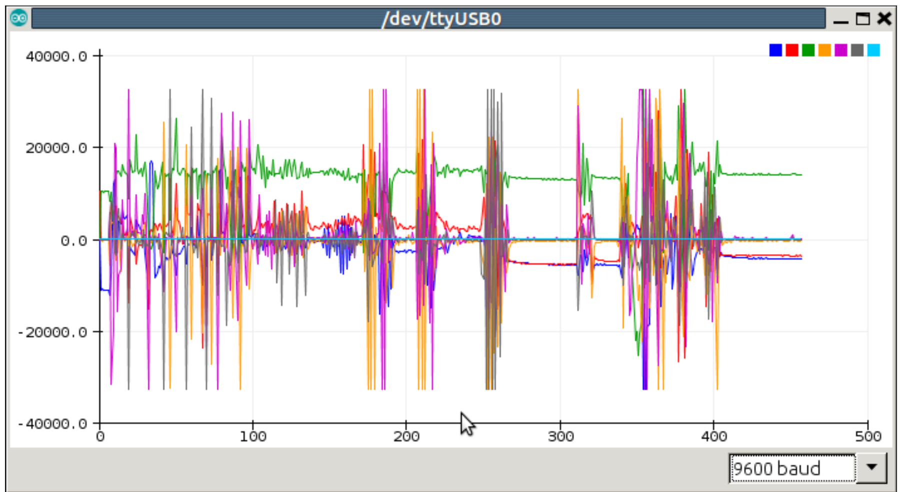


Mòdul acceleròmetre + giroscopi I2C GY-521

Aquest mòdul I2C, que pot treballar a 3,3V i és molt econòmic, ens dona la temperatura i les acceleracions lineals i angulars segons els tres eixos X, Y i Z.

A l'exemple GY-521 utilitzarem el plotter sèrie per visualitzar els canvis en aquests valors:



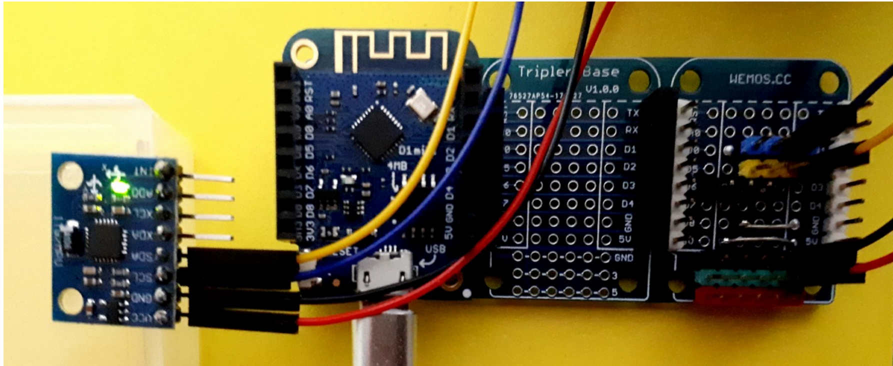



```
#include<Wire.h>
const int MPU_addr=0x68;  // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}
void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  Serial.print(AcX); Serial.print(",");
  Serial.print(AcY); Serial.print(",");
  Serial.print(AcZ); Serial.print(",");
  Serial.print(GyX); Serial.print(",");
  Serial.print(GyY); Serial.print(",");
  Serial.print(GyZ); Serial.print(",");
  Serial.println(Tmp/340.00+36.53);  //equation for temperature in degrees C
from datasheet
  delay(333);
}
```

Fixeu-vos que en aquest cas no he fet servir cap llibreria, sinó comandes I2C pures per llegir els registres del dispositiu. En realitat, les llibreries de components I2C el que fan es facilitar aquestes

comandes sota el nom d’una funció. El cor d’aquest mòdul, el MPU6050, té 118 registres!

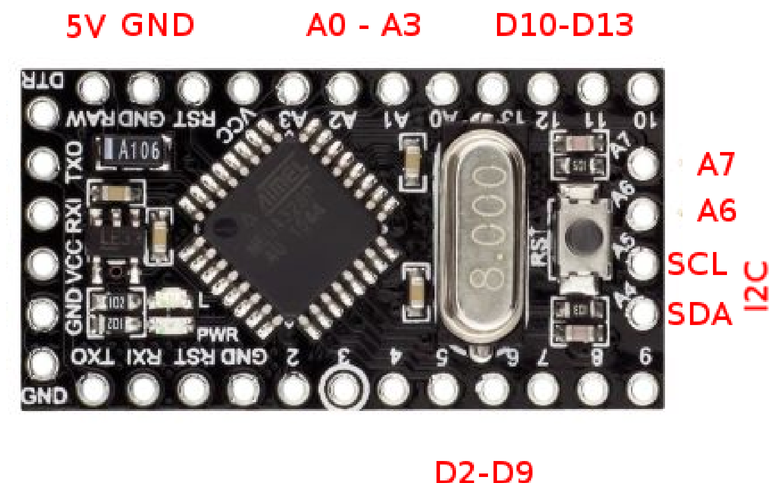
Però existeix una llibreria per a aquest mòdul. Es diu MPU6050, per si voleu provar-la.



Arduino Pro mini com a esclau I2C

Com heu vist les dues limitacions més importants de l'ESP8266 són l'existència de només una entrada analògica (i, a més a més, de petita impedància) i el reduït nombre d'entrades digitals disponibles (especialment si fem servir el bus I2C).

Per això al kit trobareu un arduino pro mini a 8 MHz i 3,3 V configurat com a esclau I2C. Amb això afegim 6 entrades analògiques (d'alta impedància) i 12 ports I/O.



Aquest mòdul funciona a 3,3 V, que pot generar amb un regulador de tensió propi. Per això el connectarem als 5 V de la placa triple (d'aquesta manera repartim el consum a 3,3 V entre els reguladors del D1 mini i del arduino pro mini).

He preparat una llibreria (I2CsapmR1) per utilitzar fàcilment aquest dispositiu. La trobareu a (https://github.com/jorts64/I2CsapmR1). Veiem l'exemple *simple* que incorpora:

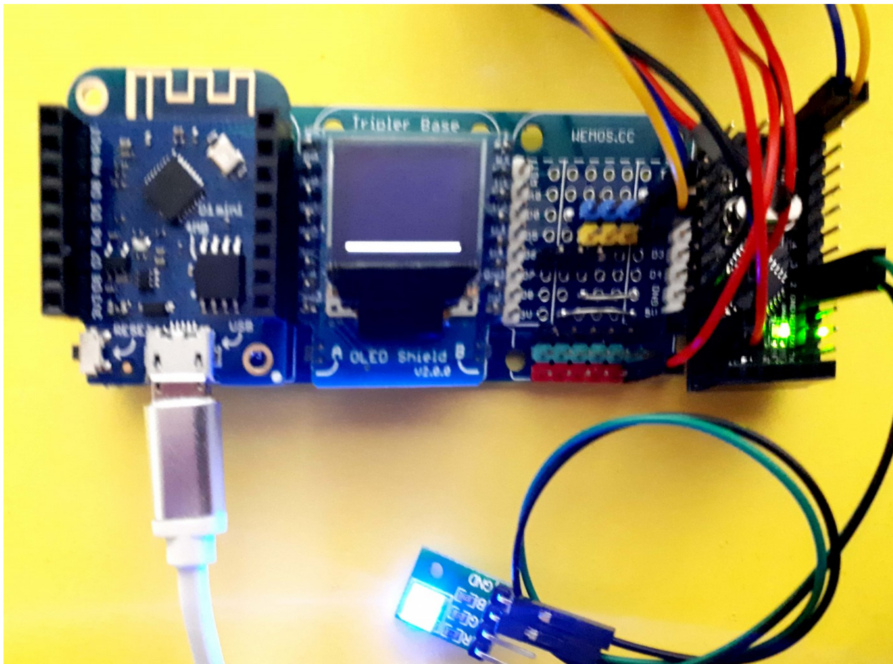
```
#include "I2CsapmR1.h"
#include <Wire.h>

I2CsapmR1 SAPM(2);

void setup() {
   Serial.begin(9600);
   SAPM.begin();
   SAPM.write(I2CsapmR1_M2,OUTPUT);
   SAPM.write(I2CsapmR1_M3,INPUT_PULLUP);
}

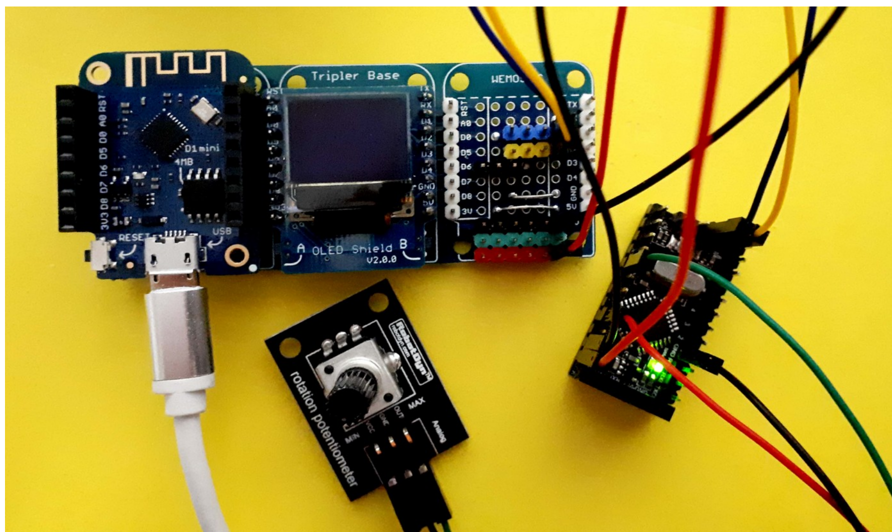
void loop() {
   SAPM.write(I2CsapmR1_D2,HIGH);
   int sensorValue = SAPM.read(I2CsapmR1_A0);
   int digValue = SAPM.read(I2CsapmR1_D3);
   Serial.print(sensorValue);
   Serial.print("  /   ");
   Serial.println(digValue);
   delay(100);
   SAPM.write(I2CsapmR1_D2,LOW);
   delay(100);
}
```

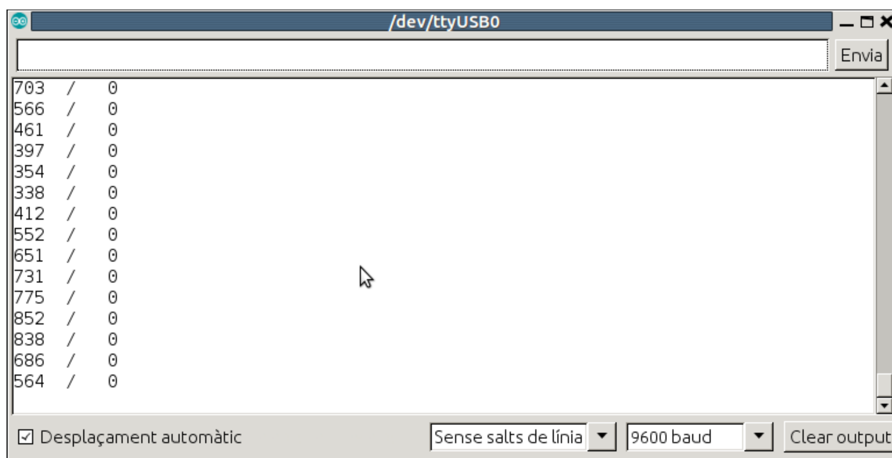
Si connectem un led al pin 2 de l’arduino pro mini veurem que fa pampallugues[12]:



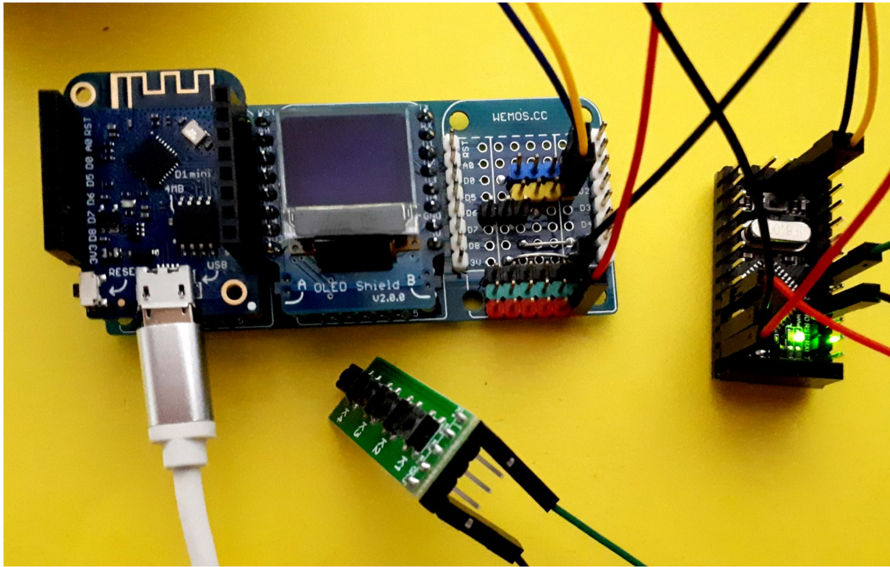
12 Encara que aquest exemple no fa servir el display OLED l’he posat per tenir al menys un dispositiu I2C al bus que posi els seus pull-ups a SCL i SDA.

Si ara connectem el potenciòmetre a l'entrada A0 de l'arduino pro mini i obrim el monitor sèrie llegirem el seu valor (primer nombre de cada línia):





Ara connectem un polsador a l'entrada 3 de l'arduino pro mini i obrim el monitor sèrie. Veurem els canvis a l'entrada (segon nombre de cada línia):





Analitzem una mica el codi:

Al setup() hem configurat les entrades digitals tal com ho faríem amb pinMode():

```
    SAPM.write(I2CsapmR1_M2,OUTPUT);
    SAPM.write(I2CsapmR1_M3,INPUT_PULLUP);
```

Al loop hem llegit les entrades digitals i analògiques i canviat les sortides digitals accedint al seu registre:

```
   SAPM.write(I2CsapmR1_D2,HIGH);
   int sensorValue = SAPM.read(I2CsapmR1_A0);
   int digValue = SAPM.read(I2CsapmR1_D3);
```

Com els registres són consecutius podem utilitzar bucles per configurar o accedir a un conjunt de pins:

```
#include "I2CsapmR1.h"
I2CsapmR1 SAPM(2);
int diginp[8];
int aninp[6];
void setup() {
  SAPM.begin();
  for (int i=I2CsapmR1_M2;i<=I2CsapmR1_M9;i++)
     SAPM.write(i,INPUT_PULLUP);
}

void loop() {
  for (int i=I2CsapmR1_M2;i<=I2CsapmR1_M9;i++)
    diginp[i-I2CsapmR1_M2]=SAPM.read(i);
  for (int i=I2CsapmR1_A0;i<=I2CsapmR1_A7;i++)
    aninp[i-I2CsapmR1_A0]=SAPM.read(i);
}
```

L’avantatge d’aquest dispositiu és el seu preu, molt econòmic, i la seva flexibilitat. Alterant el firmware[13] podem utilitzar-lo com a una navalla suïssa per a qualsevol aplicació:

- sortides amb pwm
- control de servos
- control de motors pas a pas
- lectura d’encoders rotatoris
- lectura de teclats matricials

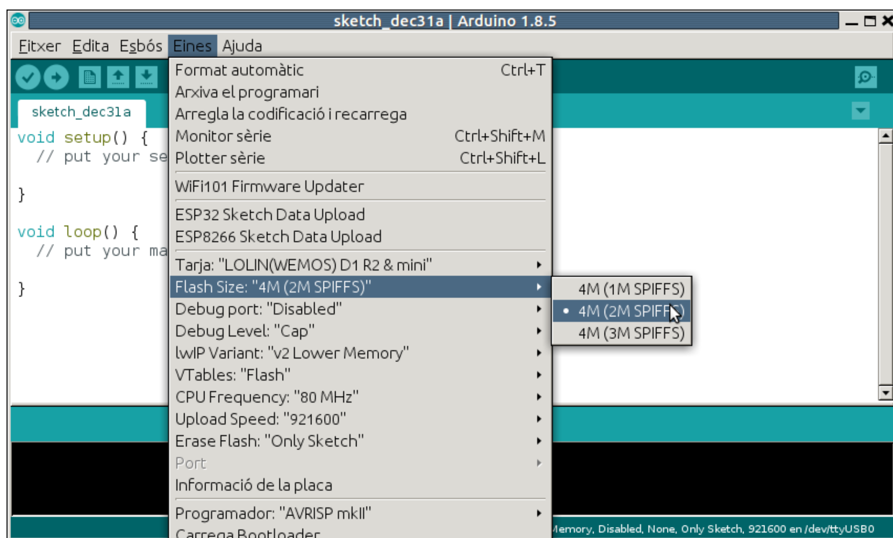
A més a més en podem tenir tants arduino pro mini al bus I2C com vulguem. Només cal canviar l’adreça I2C dels dispositius al seu firmware.

13 Veure Annex 8: Firmware de l’arduino pro mini

Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.

Una de les joies de l'ESP8266 és el seu sistema de fitxers SPIFFS, que simula una SD. Com l'arduino IDE només aprofita 1 MB de memòria i el nostre D1 mini en té 4 MB de flash, podem dedicar algun MB a aquesta SD virtual.

Per utilitzar aquesta opció cal escollir a l'IDE la mida de la SD virtual. Encara que podríem dedicar els 3 MB restants, jo prefereixo 2 MB, reservant 1 MB per l'actualització WiFi del programa (OTA)[14].



Cal crear a la carpeta del programa una subcarpeta amb el nom data, on posarem els fitxers amb que volem carregar inicialment la SD virtual. La imatge d'aquesta SD virtual l'enviarem al nostre D1 mini

14 Veure Part IX. Actualització WiFi del programa (OTA)

amb l'opció ESP8266 Sketch Data Upload[15], que haurem d'utilitzar amb el monitor sèrie tancat (si el tenim obert tindrem problemes per connectar-nos al D1 mini poer enviar aquesta imatge de la SD).

Al programa cal posar la línia

```
#include <FS.h>
```

Us pot semblar que 2 MB són molt poc, especialment tenint en compte la capacitat de les SDs actuals, però al llarg d'aquest capítol veureu que en podem treure un bon profit d'aquests 2 MB.

15 Cal instal·lar de forma separada aquesta eina. Veure Annex 5: Instal·lació Arduino per a ESP8266.

Afegint fitxers al nostre servidor web. La funció serveStatic().

De vegades l’únic que ens interessa es poder tenir fitxers a la nostra SD virtual amb els quals millorar l’aspecte de les pàgines webs generades: imatges, codi javascript o estils css, manuals en PDF ...

La funció *serveStatic()* fa justament això: quan es demana un fitxer al servidor web que comença amb un cert patró el busca a la SD virtual i l’envia directament. Així de simple.

Veiem el meu exemple *serveStatic*:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>

const char *ssid = "Nom de la xarxa";
const char *password = "contrasenya";

ESP8266WebServer server ( 80 );

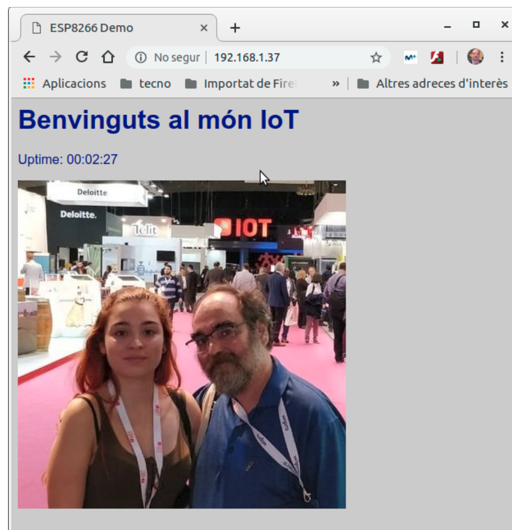
void handleRoot() {
  char temp[400];
  int sec = millis() / 1000;
  int min = sec / 60;
  int hr = min / 60;
  snprintf ( temp, 400,
"<html>\
  <head>\
    <meta http-equiv='refresh' content='5'/>\
    <title>ESP8266 Demo</title>\
    <style>\
      body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-
Serif; Color: #000088; }\
    </style>\
  </head>\
  <body>\
    <h1>Benvinguts al m&oacute;n IoT</h1>\
    <p>Uptime: %02d:%02d:%02d</p>\
    <img src='/img/IoT.jpg'>\
  </body>\
</html>",
    hr, min % 60, sec % 60
  );
  server.send ( 200, "text/html", temp );
}

void handleNotFound() {
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
```

```
  message += "\nMethod: ";
  message += ( server.method() == HTTP_GET ) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for ( uint8_t i = 0; i < server.args(); i++ ) {
    message += " " + server.argName ( i ) + ": " + server.arg ( i ) + "\n";
  }
  server.send ( 404, "text/plain", message );
}

void setup ( void ) {
  Serial.begin(9600);
  WiFi.begin ( ssid, password );
  SPIFFS.begin();
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
  }
  Serial.println ( "" );
  Serial.print ( "Connected to " );
  Serial.println ( ssid );
  Serial.print ( "IP address: " );
  Serial.println ( WiFi.localIP() );
  server.on ( "/", handleRoot );
  server.serveStatic("/img", SPIFFS, "/img"); //  imatges dir
  server.onNotFound ( handleNotFound );
  server.begin();
  Serial.println ( "HTTP server started" );
}

void loop ( void ) {
  server.handleClient();
}
```



Com podreu suposar analitzant el codi a la carpeta *data* he creat una subcarpeta *img* on he posat una fotografia amb el nom *IoT.jpg* (que

millor fotografia que una amb la meva filla Ingrid visitant la *IOTSWC18* el passat 17 d'octubre!).

Amb la línia

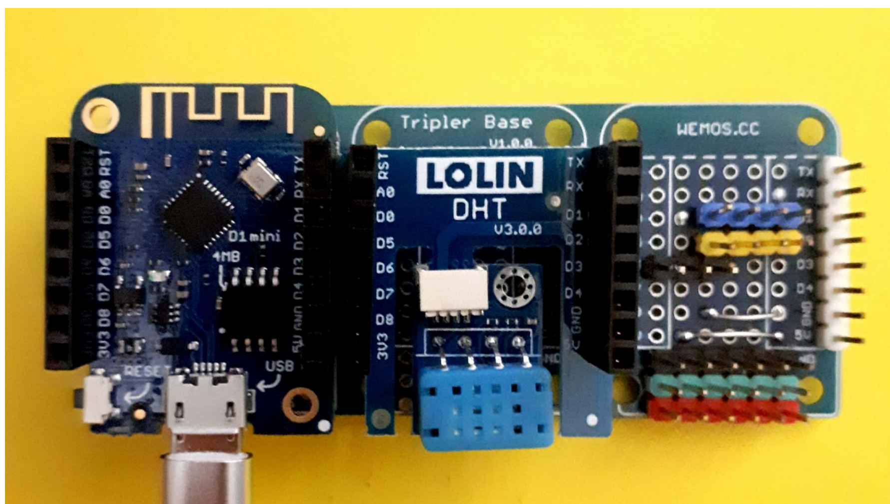
```
  server.serveStatic("/img", SPIFFS, "/img"); //  imatges dir
```

fem tota la màgia. Ara quan el servidor rep una petició d'un fitxer que comença per *img* el busca a la carpeta *img* de la SD virtual i l'envia directament.

Enregistrar dades a la SD virtual i poder recuperar-les via WiFi. Generació de fitxers compatibles amb fulls de càlcul

Aquesta es una tasca molt comú: un datalogger. Si emmagatzemem les dades a la SD en un fitxer CSV després el podrem recuperar mitjançant el servidor web i obrir-lo amb qualsevol paquet ofimàtic[16].

Veiem el meu exemple *datalogger* que utilitza el DHT shield per mesurar la temperatura:




```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>
#include <WEMOS_DHT12.h>

DHT12 dht12;
ESP8266WebServer server ( 80 );
unsigned long ara;

const char *ssid = "jortsnet";
const char *password = "9periodico";

void espera (int n) {
```

16 Si voleu un anàlisi més profund de les dades us recomano el programari QtiPlot, disponible com a GPLv2 a https://sourceforge.net/projects/qtiplot.berlios/

```
  while (millis()<ara+n) {
    delay(1);
    server.handleClient();
  }
}

void setup() {
  Serial.begin(9600);
  SPIFFS.begin();
  WiFi.begin ( ssid, password );
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
  }
  Serial.println ( "" );
  Serial.print ( "Connected to " );
  Serial.println ( ssid );
  Serial.print ( "IP address: " );
  Serial.println ( WiFi.localIP() );
  server.serveStatic("/dades", SPIFFS, "/dades"); //  gauge files dir
  server.begin();
  SPIFFS.remove("/dades/dat1.csv");
}

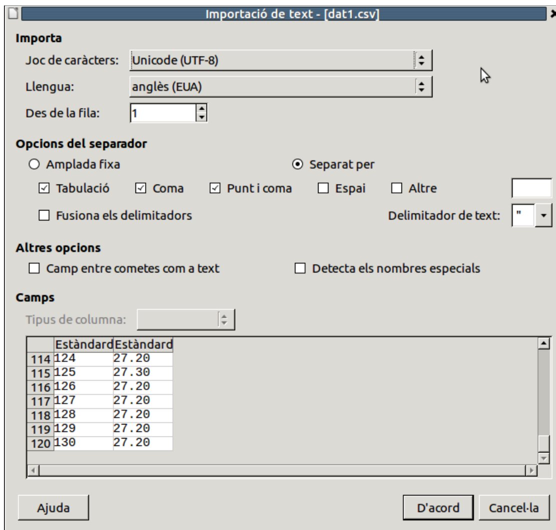
void loop() {
  File f = SPIFFS.open("/dades/dat1.csv","a");
    for(int i=1; i<=100; i++){
      ara = millis();
      if(dht12.get()==0){
        f.print(ara/1000);
        f.print(",");
        f.println(dht12.cTemp);
        espera(1000);
      }

    }
  f.close();
}
```

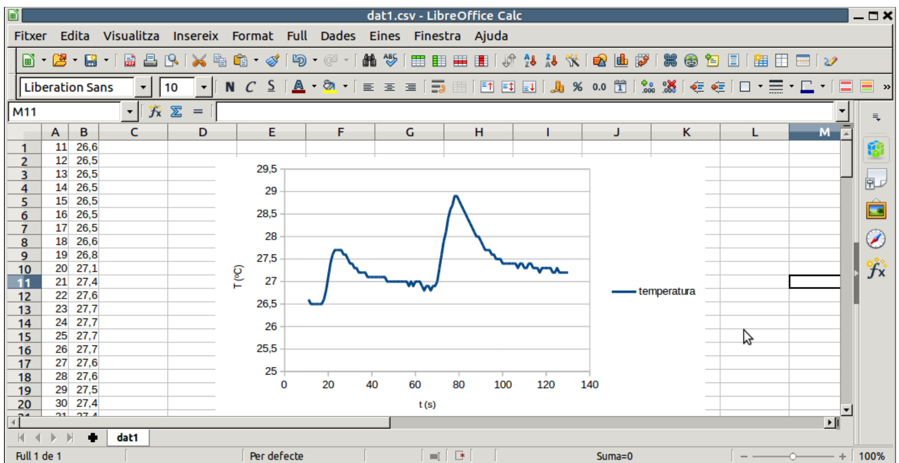
Fixeu-vos al programa com fem servir la funció *f.print()* per desar les dades a la SD virtual i com el *serveStatic()* ens permet accedir a elles via WiFi.

Una vegada enregistrades les dades podem baixar el fitxer amb un navegador apuntant a http://192.168.1.37/dades/dat1.csv (en el meu cas va agafar la IP 192.168.1.37).

Quan obrim el fitxer recomano triar idioma anglès (EUA) per interpretar correctament el separador decimal:



I ja podem treballar amb les dades:



Llegint dades de la SD virtual

La SD virtual és el lloc ideal per desar dades com ara definicions de bitmaps o fragments de pàgines web. El problema és interpretar correctament aquestes dades.

El meu exemple *bitmapSPIFFS* carreguem la definició d'un bitmap per a la pantalla OLED. Com aquest l'hem obtingut mitjançant l'eina *image2cpp*[17] la tenim en un array amb valor hexadecimals. Caldrà trobar cada element de la matriu i convertir el format de text hexadecimal:

```
#include <Wire.h>
#include <SFE_MicroOLED.h>
#include <FS.h>

#define PIN_RESET 255  //
#define DC_JUMPER 0  // I2C Addres: 0 - 0x3C, 1 - 0x3D
MicroOLED oled(PIN_RESET, DC_JUMPER);  // I2C Example
File f;
uint8_t pixels[16*24];

void setup()
{
  SPIFFS.begin();
  oled.begin();
  oled.clear(ALL);
  oled.display();
  delay(1000);
  oled.clear(PAGE);
  loadbitmap();
  oled.drawBitmap(pixels);
  oled.display();
}

void loop() {
}

String getLine() {
  String S = "" ;
  char c = f.read();
  while ( c != '\n') {
    S = S + c ;
    c = f.read();
    }
  return(S) ;
}

int convertFromHex(int ascii){
  if(ascii > 0x39) ascii -= 7; // adjust for hex letters upper or lower case
```

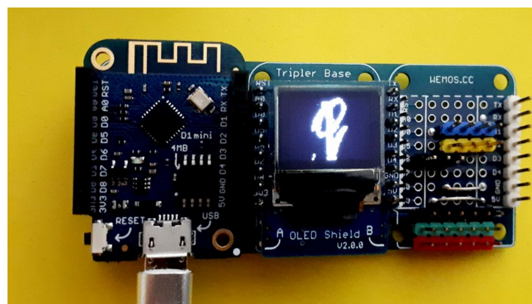
17 Disponible a http://javl.github.io/image2cpp/. Recordeu fer servir la codificació vertical i una mida de 64 x 48 px.

```
  return(ascii & 0xf);
}

void loadbitmap(){
  uint8_t sa[16];
  String txt;  int i,j,k;
  f = SPIFFS.open("/bitmap.dat","r");
    for (i=0;i<24;i++){
      txt=getLine();
      k=0;
      for (j=0; j<16; j++) {
        while (txt.charAt(k) != 'x') k++;
        sa[j] = convertFromHex(txt[k+1])*16;
        sa[j] += convertFromHex(txt[k+2]);
        k++;
      }
      for (j=0;j<16;j++) {
          pixels[i*16+j]=sa[j];
      }
    }
  f.close();
}
```

I al fitxer bitmap.dat tenim les dades:

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF8, 0x78, 0x7C, 0x1C, 0x1C, 0x1E, 0x0E, 0x0C, 0x1C, 0xF0, 0xE0, 0x80, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xE0, 0xF0, 0xFC, 0xFC, 0x7F, 0x3F, 0x1F, 0xFF, 0xFF, 0xFF,
0xFF, 0xEB, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xF0, 0x7F, 0x1F, 0x03, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC0, 0xFC, 0xFF, 0xFF, 0xFF, 0xFF, 0x9F, 0x03, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0xC6, 0xE4, 0xF0, 0xF8, 0xDE, 0x0F, 0x03, 0x01, 0x00, 0x80, 0x80, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0x01, 0x00, 0x02, 0x1F, 0x3F, 0x70, 0x58, 0x30, 0x30, 0x18, 0x38, 0xFF, 0xFF,
0xFF, 0xFF, 0x07, 0x06, 0x02, 0x80, 0xF0, 0xF0, 0xFC, 0x7E, 0x1F, 0x07, 0x03, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0xF1, 0xFC, 0xFF, 0xFF, 0x3F, 0x07, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x80, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xCF, 0xFF,
0xFE, 0xFF, 0xFF, 0xBF, 0xCF, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

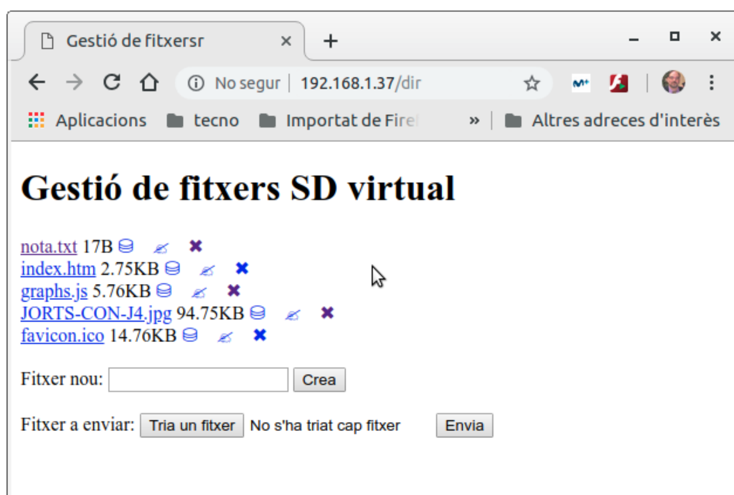


FSManager, la navalla suïssa per a la gestió de fitxers a la SD virtual via WiFi

Ja haureu vist que carregar la imatge de la SD virtual és lent. Especialment si estem modificant arxius d'estil CSS o pàgines HTML i codi javascript per millorar l'aspecte de la nostra interfície web.

El meu exemple *FSManager* es pot utilitzar com a una plantilla per als vostres programes. Es fàcil d'adaptar i millorar, i permet enviar, crear, editar, veure, baixar i esborrar tot tipus de fitxers. Això si, està limitat a gestionar un sol directori, que per defecte és l'arrel.

A més a més facilita la connexió WiFi i pot treballar tant en mode AP com STA. Només cal canviar el nom de xarxa i contrasenya, així com el tipus de connexió, si s'escau. No utilitza fitxers externs ni llibreries al núvol[18].



18 El codi de FSManager ha evolucionat al llarg de la redacció d'aquest llibre, afegint noves prestacions. Trobareu la versió més completa i potent a Annex 9: Afegint la funcionalitat de FSManager als nostres projectes

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>

#define DBG_OUTPUT_PORT Serial

const char* ssid = "esp8266AP";
const char* password = "robotica";
const char* host = "esp8266fs";
// tria el tipus de connexió. Client WIFI_STA, Access Point WIFI_AP
#define MODE_WIFI WIFI_AP

ESP8266WebServer server(80);
File fsUploadFile;

// Helper functions prototypes
void OkRetorn();
void printDirectory();
void handleFileCreate();
void handleFileDelete();
void handleFileUpload();
String formatBytes(size_t bytes);
String getContentType(String filename);
bool handleFileRead(String path);
void handleFileEdit();
void handleFileSave();
void initHelper();
void initWifi();
void pageHead();

void setup(void) {
  DBG_OUTPUT_PORT.begin(115200);
  SPIFFS.begin();

  initHelper(); //inclou connexió Wifi

  //get heap status, analog input value and all GPIO statuses in one json call
  server.on("/all", HTTP_GET, []() {
    String json = "{";
    json += "\"heap\":" + String(ESP.getFreeHeap());
    json += ", \"analog\":" + String(analogRead(A0));
    json += ", \"gpio\":" + String((uint32_t)(((GPI | GPO) & 0xFFFF) | ((GP16I
& 0x01) << 16)));
    json += "}";
    server.send(200, "text/json", json);
    json = String();
  });

  server.begin();
  DBG_OUTPUT_PORT.println("HTTP server started");
}

void loop(void) {
  server.handleClient();
}


//----------------------------------------- Helper functions
---------------------------

void pageHead(){
  server.setContentLength(CONTENT_LENGTH_UNKNOWN);
  server.send(200, "text/html", "");
```

```
  server.sendContent("<!DOCTYPE html><html><head><title>Gesti&oacute; de
fitxersr</title><meta charset='UTF-8'><meta name='viewport'
content='width=device-width'/></head><body>");
}

void initWifi(){
  if (MODE_WIFI==WIFI_STA) {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    DBG_OUTPUT_PORT.printf("Connecting to %s\n", ssid);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      DBG_OUTPUT_PORT.print(".");
    }
    DBG_OUTPUT_PORT.println("");
    DBG_OUTPUT_PORT.print("Connected! IP address: ");
    DBG_OUTPUT_PORT.println(WiFi.localIP());

    DBG_OUTPUT_PORT.print("Open http://");
    DBG_OUTPUT_PORT.print(WiFi.localIP());
    DBG_OUTPUT_PORT.println("/dir to see the file browser");
  }
  else {
    WiFi.mode(WIFI_AP);
    WiFi.softAP(ssid, password);
    DBG_OUTPUT_PORT.print("Open http://192.168.4.1/dir to see the file
browser");
  }
}

void initHelper(){
  DBG_OUTPUT_PORT.print("\n");
  DBG_OUTPUT_PORT.setDebugOutput(true);
    Dir dir = SPIFFS.openDir("/");
    while (dir.next()) {
      String fileName = dir.fileName();
      size_t fileSize = dir.fileSize();
      DBG_OUTPUT_PORT.printf("FS File: %s, size: %s\n", fileName.c_str(),
formatBytes(fileSize).c_str());
      DBG_OUTPUT_PORT.printf("\n");
  }

  initWifi();

  //SERVER INIT
  server.on("/dir", HTTP_GET, printDirectory);
  server.on("/create", HTTP_GET, handleFileCreate);
  server.on("/delete", HTTP_GET, handleFileDelete);
  //first callback is called after the request has ended with all parsed
arguments
  //second callback handles file uploads at that location
  server.on("/upload", HTTP_POST, []() {
      OkRetorn();
  }, handleFileUpload);
  server.on("/edit", HTTP_GET, handleFileEdit);
  server.on("/save", HTTP_POST, handleFileSave);
  //called when the url is not defined here
  //use it to load content from SPIFFS
  server.onNotFound([]() {
    if (!handleFileRead(server.uri())) {
      server.send(404, "text/plain", "FileNotFound");
    }
  });
```

```
}

void OkRetorn() {
  pageHead();
  server.sendContent("Operaci&oacute; realitzada amb &egrave;xit<br/><a
href='/dir'>Tornar a gesti&oacute; de fitxers</a></body></html>");
}

void printDirectory(){
  Dir dir = SPIFFS.openDir("/");
  pageHead();
  server.sendContent("<h1>Gesti&oacute; de fitxers SD virtual</h1>");
  while (dir.next()) {
    String fileName = dir.fileName();
    size_t fileSize = dir.fileSize();
    DBG_OUTPUT_PORT.printf("FS File: %s, size: %s\n", fileName.c_str(),
formatBytes(fileSize).c_str());
    String output;
    output += "<a href='/";
    output += fileName.substring(1);
    output +="'>";
    output += fileName.substring(1);
    output +="</a> ";
    output += formatBytes(fileSize).c_str();
    output += "     <a href='";
    output += fileName;
    output +="?download=1' style='text-decoration: none;'> &#9921;
</a>  ";
    output += "     <a href='/edit?fe=";
    output += fileName;
    output +="' style='text-decoration: none;'> &#9997; </a>  ";
    output += "     <a href='/delete?killfitxer=";
    output += fileName;
    output +="' style='text-decoration: none;'> &#10006; </a><br/>";
    server.sendContent(output);
    DBG_OUTPUT_PORT.println(output);

  }

  server.sendContent("<br/><form action='/create'>Fitxer nou: <input
type='text' name='noufitxer' value=''>  <input type='submit'
value='Crea'></form>");
  server.sendContent("<br/><form method='post' enctype='multipart/form-data'
action='/upload'>Fitxer a enviar: <input type='file' name='myFile'><input
type='submit' value='Envia'></form>");
  server.sendContent("</body></html>");
}

void handleFileCreate() {
  if (server.args() == 0) {
    return server.send(500, "text/plain", "BAD ARGS");
  }
  String path = "/" + server.arg("noufitxer");
  DBG_OUTPUT_PORT.println("handleFileCreate: " + path);
  if (path == "/") {
    return server.send(500, "text/plain", "BAD PATH");
  }
  if (SPIFFS.exists(path)) {
    return server.send(500, "text/plain", "FILE EXISTS");
  }
  File file = SPIFFS.open(path, "w");
  if (file) {
    file.close();
  } else {
```

```
    return server.send(500, "text/plain", "CREATE FAILED");
  }
  OkRetorn();
  path = String();
}

void handleFileDelete() {
  if (server.args() == 0) {
    return server.send(500, "text/plain", "BAD ARGS");
  }
  String path = server.arg("killfitxer");
  DBG_OUTPUT_PORT.println("handleFileDelete: " + path);
  if (path == "/") {
    return server.send(500, "text/plain", "BAD PATH");
  }
  if (!SPIFFS.exists(path)) {
    return server.send(404, "text/plain", "FileNotFound");
  }
  SPIFFS.remove(path);
  OkRetorn();
  path = String();
}

void handleFileUpload() {
  if (server.uri() != "/upload") {
    return;
  }
  HTTPUpload& upload = server.upload();
  if (upload.status == UPLOAD_FILE_START) {
    String filename = upload.filename;
    if (!filename.startsWith("/")) {
      filename = "/" + filename;
    }
    DBG_OUTPUT_PORT.print("handleFileUpload Name: ");
DBG_OUTPUT_PORT.println(filename);
    fsUploadFile = SPIFFS.open(filename, "w");
    filename = String();
  } else if (upload.status == UPLOAD_FILE_WRITE) {
    //DBG_OUTPUT_PORT.print("handleFileUpload Data: ");
DBG_OUTPUT_PORT.println(upload.currentSize);
    if (fsUploadFile) {
      fsUploadFile.write(upload.buf, upload.currentSize);
    }
  } else if (upload.status == UPLOAD_FILE_END) {
    if (fsUploadFile) {
      fsUploadFile.close();
    }
    DBG_OUTPUT_PORT.print("handleFileUpload Size: ");
DBG_OUTPUT_PORT.println(upload.totalSize);
  }
}

String formatBytes(size_t bytes) {
  if (bytes < 1024) {
    return String(bytes) + "B";
  } else if (bytes < (1024 * 1024)) {
    return String(bytes / 1024.0) + "KB";
  } else if (bytes < (1024 * 1024 * 1024)) {
    return String(bytes / 1024.0 / 1024.0) + "MB";
  } else {
    return String(bytes / 1024.0 / 1024.0 / 1024.0) + "GB";
  }
}
```

```
String getContentType(String filename) {
  if (server.hasArg("download")) {
    return "application/octet-stream";
  } else if (filename.endsWith(".htm")) {
    return "text/html";
  } else if (filename.endsWith(".html")) {
    return "text/html";
  } else if (filename.endsWith(".css")) {
    return "text/css";
  } else if (filename.endsWith(".js")) {
    return "application/javascript";
  } else if (filename.endsWith(".png")) {
    return "image/png";
  } else if (filename.endsWith(".gif")) {
    return "image/gif";
  } else if (filename.endsWith(".jpg")) {
    return "image/jpeg";
  } else if (filename.endsWith(".ico")) {
    return "image/x-icon";
  } else if (filename.endsWith(".xml")) {
    return "text/xml";
  } else if (filename.endsWith(".pdf")) {
    return "application/x-pdf";
  } else if (filename.endsWith(".zip")) {
    return "application/x-zip";
  } else if (filename.endsWith(".gz")) {
    return "application/x-gzip";
  }
  return "text/plain";
}

bool handleFileRead(String path) {
  DBG_OUTPUT_PORT.println("handleFileRead: " + path);
  if (path.endsWith("/")) {
    path += "index.htm";
  }
  String contentType = getContentType(path);
  String pathWithGz = path + ".gz";
  if (SPIFFS.exists(pathWithGz) || SPIFFS.exists(path)) {
    if (SPIFFS.exists(pathWithGz)) {
      path += ".gz";
    }
    File file = SPIFFS.open(path, "r");
    server.streamFile(file, contentType);
    file.close();
    return true;
  }
  return false;
}

void handleFileEdit() {
  if (server.args() == 0) {
    return server.send(500, "text/plain", "BAD ARGS");
  }
  String path = server.arg("fe");
  DBG_OUTPUT_PORT.println("handleFileEdit: " + path);
  if (path == "/") {
    return server.send(500, "text/plain", "BAD PATH");
  }
  if (!SPIFFS.exists(path)) {
    DBG_OUTPUT_PORT.print("handleFileEdit Name: ");
DBG_OUTPUT_PORT.print(path);DBG_OUTPUT_PORT.println(" FileNotFound");
    return server.send(404, "text/plain", "FileNotFound");
  }
```

```
  pageHead();
  server.sendContent("<form action='/save' method='POST'>Fitxer: <input 
type='text' name='nomfitxer' value='");
  server.sendContent(path);
  server.sendContent("' readonly><br/><textarea name='cos' rows='30' cols='40' 
wrap='off'>");

  File file = SPIFFS.open(path,"r");
  char buffer[2];
  buffer[1] = 0;
//  server.streamFile(file, "text/plain");
  while (file.readBytes(buffer, 1)!=0 ) {
    server.sendContent(buffer);
  }
  file.close();
  server.sendContent("</textarea><br><input type='submit' 
value='Desa'></form></body></html>");

  path = String();
}

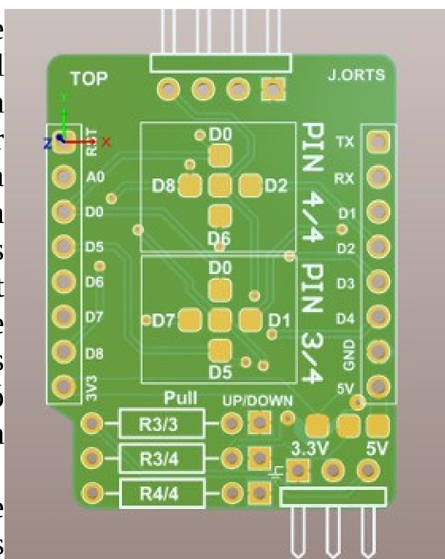
void handleFileSave() {
  String path = server.arg("nomfitxer");
  String text = server.arg("cos");
  DBG_OUTPUT_PORT.println("handleFileSave: " + path);
  DBG_OUTPUT_PORT.println("text: " + text);
  File file = SPIFFS.open(path,"w");
  if (!file) {
    DBG_OUTPUT_PORT.println("file open failed");
  }
  file.print(text);
  file.close();
  OkRetorn();
  path = String();
  text = String();
}
```

Part V. Altres shields D1 mini

El nostre kit té un bon grapat de shields, però no els té tots. És interessant conèixer la resta per seleccionar el maquinari d’un projecte. Si trobem un shield amb el component que necessitem tindrem un disseny més compacte.

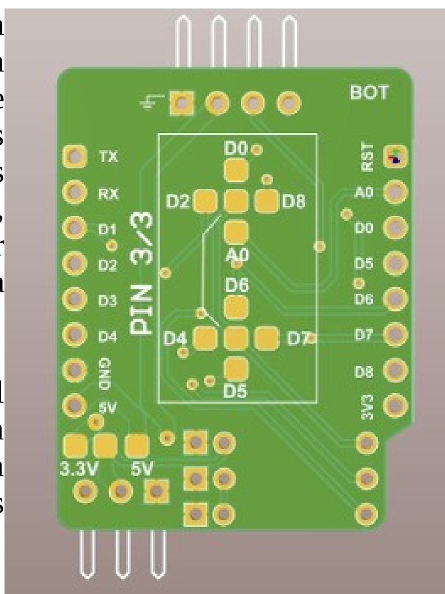
A més a més, aquest tema és una excusa perfecta per mostrar-vos altres dispositius i tècniques de programació.

Shield de connexions (JORTS)



A l'hora de passar del prototip de desenvolupament o viabilitat al prototip de comercialització un dels problemes que em vaig trobar al llarg del curs 2017-18 va ser la connexió de maquinari extern. La placa triple amb el seus pins mascle al 3r bloc facilita molt aquesta connexió al prototip de desenvolupament, en el qual els alumnes comproven la distribució correcta de pins i desenvolupen els programes de control.

Però al passar al prototip de comercialització, on els alumnes minimitzen el cost del producte i dissenyen una capsa amb la impressora 3D, l'única solució a l'abast era l'ús de plaques de prototips per posar els connectors al maquinari extern. Aquesta és una solució lenta i, de vegades, complicada, ja que cal dissenyar aquesta placa i soldar-la correctament.



Per tot això a l'estiu del 2018 el meu nebot Javier i jo vam dissenyar i enviar a fabricar a Xina un shield de connexions configurable.

A la cara superior (TOP) podem configurar el connector J4 de 4 pins, i a la cara inferior (BOT) el J3 de 3 pins. Per a cada senyal podem, a la cara superior, afegir una resistència de pull-up o pull-down.

Connector J4 (TOP)
pin 1: GND (indicat amb un quadrat)
pin 2: VCC. Cal fer un pont entre 3.3V o 5V i el pad central per seleccionar aquest voltatge
pin 3: podem escollir entre D0, D1, D5 o D7. Cal fer un pont entre el pad corresponent i el pad central
pin 4: podem escoliir entre D0, D2, D6 o D8. Cal fer un pont entre el pad corresponent i el pad central

A la cara superior (TOP) podem posar una resistència per a cada senyal si ens interessa fixar un valor per defecte a aquest pin.

- Si la resistència arriba al forat indicat per un quadrat serà un pull-down (valor per defecte 0V)
- Si la resistència és una mica més curta i connecta al forat indicat per un cercle serà un pull-up (valor per defecte VCC per a aquest connector)

Connector J3 (BOT)
pin 1: GND (indicat amb un quadrat)
pin 2: VCC. Cal fer un pont entre 3.3V o 5V i el pad central per seleccionar aquest voltatge
pin 3: podem escollir entre A0, D0, D2, D4, D5, D6, D7 o D8. Cal fer un pont entre el pad corresponent i el pad central

A continuació podeu veure un exemple real d'aplicació. Uns alumnes meus necessitaven per a un prototip de comercialització del seu treball de recerca un connector per a l'entrada analògica A0. Amb aquest shield el van tenir funcionant en 5 minuts:

Pel desenvolupament de prototips comercials amb el nostre kit utilitzarem 2 configuracions típiques, CON1 i CON2.

Shield CON1

Caldrà fer alguns ponts a la placa, de forma que

- J4 té connexions a D5 (3/4) i D0 (4/4) amb VCC = 3,3V (2/4)
- J3 té connexió a D6 (3/3) i VCC = 5 V (2/3)

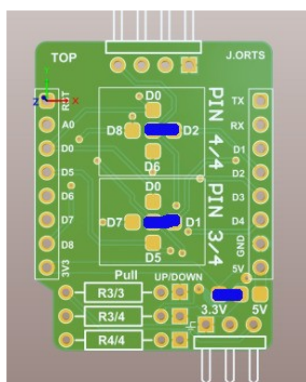
D'aquesta manera podem connectar un servo a J3 i mòduls amb entrades o sortides digitals a J4. S'han escollit D5 i D0 per a J4 per la seva baixa col·lisió amb altres shields en els projectes considerats.

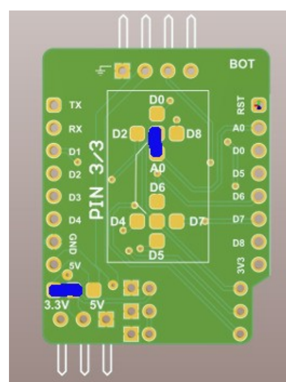
Shield CON2

Caldrà fer alguns ponts a la placa, de forma que

- J4 té connexions a D1 (3/4) i D2 (4/4) amb VCC = 3,3V (2/4)
- J3 té connexió a A0 (3/3) i VCC = 3,3 V (2/3)

D'aquesta manera podem connectar mòduls I2C a J4 (SCL/D1 a 3/4 i SDA/D2 a 4/4, 3,3V a 2/4, GND a 1/4) i sensors analògics a J3 (A0 a 3/3, 3,3V a 2/3, GND a 1/3)

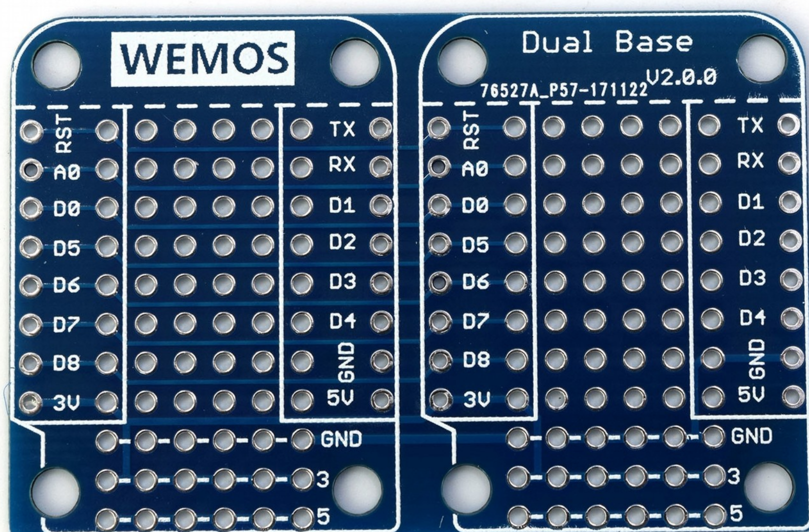




Dual base

Amb només 2 columnes de shields, aquesta base és molt útil per prototips comercials.

L'antiga versió només portava els connectors per a cada columna. L'actual (V2.0.0) és semblant a la triple base, amb línies de potència i zona de prototipatge.

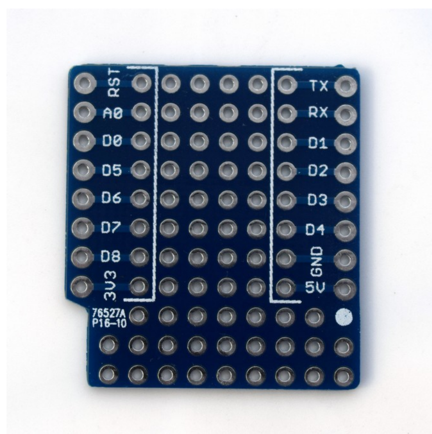


Amb la nova versió podem reproduir l'estructura de la nostra 3a columna per a projectes amb nombrosos components externs, o incloure algun d'ells a la zona de prototipatge.

Protoboard shield

Molt útil per afegir altres components o connectors.
La primera i darrera columna de topos estan connectades a les respectives senyals del bus.

La versió antiga del kit (abril 2018) portava un semàfor de dos leds fet amb aquest shield.



WS2812B RGB Shield

Aquest mòdul porta un led RGB WS2812B, que fa servir un protocol d'un sol fil (connectat a D2).
Malauradament D2 s'utilitza típicament com SDA al bus I2C. Es a dir, si utilitzem aquest mòdul, no podrem fer servir cap shield amb I2C.
Aquest shield estava inclòs a la versió antiga del kit (abril 2018), i ha estat substituït pel RGB shield, amb 7 leds i connexió configurable.



Aquest shield el podem fer servir per a wearables, on podem canviar amb el fil conductor el pin on va connectat i tenir-ne diversos leds connectats, o en aplicacions on la resta de mòduls no utilitzen I2C o només s'utilitza aquest shield. Un exemple d'aquest darrer cas és el m,eu exemple *peana*, on il·luminem amb diversos colors una escultura feta amb impressora 3D des de la base amb aquest shield:

```
//Install [Adafruit_NeoPixel_Library]
(https://github.com/adafruit/Adafruit_NeoPixel) first.

#include <Adafruit_NeoPixel.h>

#define PIN            D2

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(1, PIN, NEO_BRG + NEO_KHZ800);

// colors RGB values from Viquipedia: https://ca.wikipedia.org/wiki/Colors_HTML

uint32_t color[]={
  0xFFC0CB, //   0 Pink
  0xFFB6C1, //   1 LightPink
  0xFF69B4, //   2 HotPink
  0xFF1493, //   3 DeepPink
  0xDB7093, //   4 PaleVioletRed
  0xC71585, //   5 MediumVioletRed
  0xFFA07A, //   6 LightSalmon
  0xFA8072, //   7 Salmon
  0xE9967A, //   8 DarkSalmon
```

```
  0xF08080, //   9 LightCoral
  0xCD5C5C, //  10 IndianRed
  0xDC143C, //  11 Crimson
  0xB22222, //  12 FireBrick
  0x8B0000, //  13 DarkRed
  0xFF0000, //  14 Red
  0xFF4500, //  15 OrangeRed
  0xFF6347, //  16 Tomato
  0xFF7F50, //  17 Coral
  0xFF8C00, //  18 DarkOrange
  0xFFA500, //  19 Orange
  0xFFD700, //  20 Gold
  0xFFFF00, //  21 Yellow
  0xFFFFE0, //  22 LightYellow
  0xFFFACD, //  23 LemonChiffon
  0xFAFAD2, //  24 LightGoldenrodYellow
  0xFFEFD5, //  25 PapayaWhip
  0xFFE4B5, //  26 Moccasin
  0xFFDAB9, //  27 PeachPuff
  0xEEE8AA, //  28 PaleGoldenrod
  0xF0E68C, //  29 Khaki
  0xBDB76B, //  30 DarkKhaki
  0xFFF8DC, //  31 Cornsilk
  0xFFEBCD, //  32 BlanchedAlmond
  0xFFE4C4, //  33 Bisque
  0xFFDEAD, //  34 NavajoWhite
  0xF5DEB3, //  35 Wheat
  0xDEB887, //  36 BurlyWood
  0xD2B48C, //  37 Tan
  0xBC8F8F, //  38 RosyBrown
  0xF4A460, //  39 SandyBrown
  0xDAA520, //  40 Goldenrod
  0xB8860B, //  41 DarkGoldenrod
  0xCD853F, //  42 Peru
  0xD2691E, //  43 Chocolate
  0x8B4513, //  44 SaddleBrown
  0xA0522D, //  45 Sienna
  0xA52A2A, //  46 Brown
  0x800000, //  47 Maroon
  0x556B2F, //  48 DarkOliveGreen
  0x808000, //  49 Olive
  0x6B8E23, //  50 OliveDrab
  0x9ACD32, //  51 YellowGreen
  0x32CD32, //  52 LimeGreen
  0x00FF00, //  53 Lime
  0x7CFC00, //  54 LawnGreen
  0x7FFF00, //  55 Chartreuse
  0xADFF2F, //  56 GreenYellow
  0x00FF7F, //  57 SpringGreen
  0x00FA9A, //  58 MediumSpringGreen
  0x90EE90, //  59 LightGreen
  0x98FB98, //  60 PaleGreen
  0x8FBC8F, //  61 DarkSeaGreen
  0x3CB371, //  62 MediumSeaGreen
  0x2E8B57, //  63 SeaGreen
  0x228B22, //  64 ForestGreen
  0x008000, //  65 Green
  0x006400, //  66 DarkGreen
  0x66CDAA, //  67 MediumAquamarine
  0x00FFFF, //  68 Aqua
  0x00FFFF, //  69 Cyan
  0xE0FFFF, //  70 LightCyan
  0xAFEEEE, //  71 PaleTurquoise
  0x7FFFD4, //  72 Aquamarine
```

```
  0x40E0D0, //  73 Turquoise
  0x48D1CC, //  74 MediumTurquoise
  0x00CED1, //  75 DarkTurquoise
  0x20B2AA, //  76 LightSeaGreen
  0x5F9EA0, //  77 CadetBlue
  0x008B8B, //  78 DarkCyan
  0x008080, //  79 Teal
  0xB0C4DE, //  80 LightSteelBlue
  0xB0E0E6, //  81 PowderBlue
  0xADD8E6, //  82 LightBlue
  0x87CEEB, //  83 SkyBlue
  0x87CEFA, //  84 LightSkyBlue
  0x00BFFF, //  85 DeepSkyBlue
  0x1E90FF, //  86 DodgerBlue
  0x6495ED, //  87 CornflowerBlue
  0x4682B4, //  88 SteelBlue
  0x4169E1, //  89 RoyalBlue
  0x0000FF, //  90 Blue
  0x0000CD, //  91 MediumBlue
  0x00008B, //  92 DarkBlue
  0x000080, //  93 Navy
  0x191970, //  94 MidnightBlue
  0xE6E6FA, //  95 Lavender
  0xD8BFD8, //  96 Thistle
  0xDDA0DD, //  97 Plum
  0xEE82EE, //  98 Violet
  0xDA70D6, //  99 Orchid
  0xFF00FF, // 100 Magenta
  0xBA55D3, // 101 MediumOrchid
  0x9370DB, // 102 MediumPurple
  0x8A2BE2, // 103 BlueViolet
  0x9400D3, // 104 DarkViolet
  0x9932CC, // 105 DarkOrchid
  0x8B008B, // 106 DarkMagenta
  0x800080, // 107 Purple
  0x4B0082, // 108 Indigo
  0x483D8B, // 109 DarkSlateBlue
  0x6A5ACD, // 110 SlateBlue
  0x7B68EE, // 111 MediumSlateBlue
  0xFFFFFF, // 112 White
  0xF5F5DC, // 113 Beige
  0xFAEBD7, // 114 AntiqueWhite
  0xFFE4E1, // 115 MistyRose
  0x808080, // 116 Gray
  0x708090  // 117 SlateGray
};

int indexcolor;
uint32_t oldcolor, newcolor;

void setup() {
  pixels.begin(); // This initializes the NeoPixel library.
  indexcolor = random(118);
  oldcolor=color[indexcolor];
  pixels.setPixelColor(0, oldcolor);
  pixels.show(); // This sends the updated pixel color to the hardware.
  delay(400); // Delay for a period of time (in milliseconds).
}

void loop() {

  indexcolor = random(118);
  newcolor=color[indexcolor];
  while (oldcolor != newcolor) {
```

```
    uint32_t oldR = oldcolor & 0xFF0000;
    uint32_t newR = newcolor & 0xFF0000;
    uint32_t oldG = oldcolor & 0x00FF00;
    uint32_t newG = newcolor & 0x00FF00;
    uint32_t oldB = oldcolor & 0x0000FF;
    uint32_t newB = newcolor & 0x0000FF;
    if (newR > oldR) {
      oldcolor += 0x10000;
    }
    if (newR < oldR) {
      oldcolor -= 0x10000;
    }
    if (newG > oldG) {
      oldcolor += 0x100;
    }
    if (newG < oldG) {
      oldcolor -= 0x100;
    }
    if (newB > oldB) {
      oldcolor += 0x1;
    }
    if (newB < oldB) {
      oldcolor -= 0x1;
    }

    pixels.setPixelColor(0, oldcolor);
    pixels.show(); // This sends the updated pixel color to the hardware.
    delay(10); // Delay for a period of time (in milliseconds).
  }
  pixels.setPixelColor(0, oldcolor);
  pixels.show(); // This sends the updated pixel color to the hardware.
  delay(1000); // Delay for a period of time (in milliseconds).
}
```

Micro SD card shield



Aquest shield ens permet treballar amb SDs reals, de forma molt semblant a com hem utilitzat la SD virtuals del sistema de fitxers SPIFFS.

Si en fet meravelles amb els 2 MB de les SDs virtuals, imagineu-vos amb els GB de les SDs reals.

Malauradament la funció *serveStatic()* no funciona amb SDs reals. Haurem de crear codi per servir nosaltres els fitxers.

L' exemple *SdserverAP* ens permet llistar els fitxers d'un directori i accedir a qualsevol fitxer de la SD. Fixeu-vos que tota la màgia del codi està en la funció *handleNotFound()*:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <SPI.h>
#include <SD.h>

#define DBG_OUTPUT_PORT Serial

const char* ssid = "ESPserver";
const char* password = "robotica";

ESP8266WebServer server(80);

static bool hasSD = false;
File uploadFile;


void returnOK() {
  server.send(200, "text/plain", "");
}

void returnFail(String msg) {
  server.send(500, "text/plain", msg + "\r\n");
}

bool loadFromSdCard(String path) {
  String dataType = "text/plain";
  if (path.endsWith("/")) {
    path += "index.htm";
```

```
  }

  if (path.endsWith(".src")) {
    path = path.substring(0, path.lastIndexOf("."));
  } else if (path.endsWith(".htm")) {
    dataType = "text/html";
  } else if (path.endsWith(".css")) {
    dataType = "text/css";
  } else if (path.endsWith(".js")) {
    dataType = "application/javascript";
  } else if (path.endsWith(".png")) {
    dataType = "image/png";
  } else if (path.endsWith(".gif")) {
    dataType = "image/gif";
  } else if (path.endsWith(".jpg")) {
    dataType = "image/jpeg";
  } else if (path.endsWith(".ico")) {
    dataType = "image/x-icon";
  } else if (path.endsWith(".xml")) {
    dataType = "text/xml";
  } else if (path.endsWith(".pdf")) {
    dataType = "application/pdf";
  } else if (path.endsWith(".zip")) {
    dataType = "application/zip";
  }

  File dataFile = SD.open(path.c_str());
  if (dataFile.isDirectory()) {
    path += "/index.htm";
    dataType = "text/html";
    dataFile = SD.open(path.c_str());
  }

  if (!dataFile) {
    return false;
  }

  if (server.hasArg("download")) {
    dataType = "application/octet-stream";
  }

  if (server.streamFile(dataFile, dataType) != dataFile.size()) {
    DBG_OUTPUT_PORT.println("Sent less data than expected!");
  }

  dataFile.close();
  return true;
}

void handleNotFound() {
  if (hasSD && loadFromSdCard(server.uri())) {
    return;
  }
  String message = "SDCARD Not Detected\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) {
    message += " NAME:" + server.argName(i) + "\n VALUE:" + server.arg(i) +
"\n";
```

```
  }
  server.send(404, "text/plain", message);
  DBG_OUTPUT_PORT.print(message);
}

void printDirectory(){
  File dir = SD.open("/files");
  dir.rewindDirectory();
  server.setContentLength(CONTENT_LENGTH_UNKNOWN);
  server.send(200, "text/html", "");
  server.sendContent("<html><head></head><body>");
  while(true) {
     File entry =  dir.openNextFile();
     if (! entry) {
       // no more files
       break;
     }
    String output;
    output += "<a href='/files/";
    output += entry.name();
    output +="'>";
    output += entry.name();
    output +="</a><br/>";
    server.sendContent(output);
    entry.close();
   }
  server.sendContent("</body></html>");
  dir.close();
}

void setup(void) {
  DBG_OUTPUT_PORT.begin(115200);
  DBG_OUTPUT_PORT.setDebugOutput(true);
  DBG_OUTPUT_PORT.print("\n");
    WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);

  server.on("/list", HTTP_GET, printDirectory);
  server.onNotFound(handleNotFound);

  server.begin();
  DBG_OUTPUT_PORT.println("HTTP server started");

  if (SD.begin(SS)) {
    DBG_OUTPUT_PORT.println("SD Card initialized.");
    hasSD = true;
  }
}

void loop(void) {
  server.handleClient();
}
```

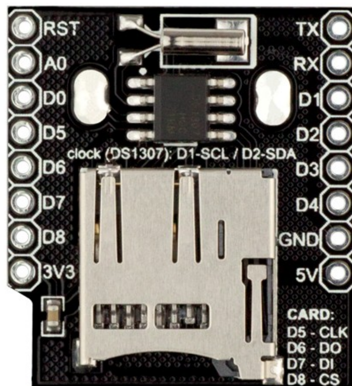
Datalogger shield

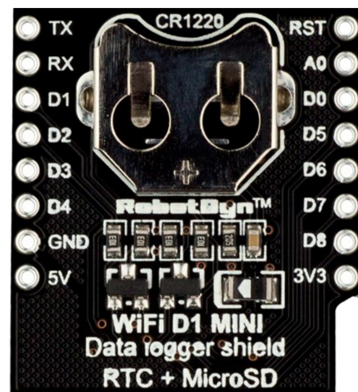
Aquest mòdul, comercialitzat pel fabricant RobotDyn, combina el lector microSD amb el RTC, facilitant molt el disseny de dataloggers.

Encara que inclòs al kit inicial d'abril de 2018, es va substituir pel RTC shield. El problema és l'alt nombre de pins utilitzats. A més a més del bus I2C (D1 i D2) pel RTC, utilitza quatre pins més (D5, D6, D7 i D8) per la SD. Això podria provocar col·lisions amb altres mòduls del kit.

És a dir, només ens queden lliures els pins D3 i D4, A0, i el bus I2C.

Per tant és un mòdul interessant quan fem servir sensors I2C, o ampliem el nombre d'entrades amb un dispositiu I2C (com l'arduino pro mini inclòs al kit).





A l'exemple *SDdatalogger* fem servir aquest shield per enregistrar les dades de temperatura d'un shield DHT juntament amb l'hora. Mitjançant un servidor web que s'activa amb el polsador del 1-button shield. Cal tenir paciència: potser caldrà esperar més de 30 s polsant el botó per engegar el servidor!

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <SD.h>
#include <WEMOS_DHT12.h>
#include <Wire.h>
```

```
#include "RTClib.h"

RTC_DS1307 RTC;
DateTime now;
DHT12 dht12;
ESP8266WebServer server ( 80 );
unsigned long ara;

const char *ssid = "jortsnet";
const char *password = "9periodico";

void espera (int n) {
  while (millis()<ara+n) {
    delay(1);
  }
}

void sendFile() {

  File dataFile = SD.open("/dat1.csv");
  server.streamFile(dataFile, "application/octet-stream");
  dataFile.close();
}

void setup() {
  pinMode(D3,INPUT);
  pinMode(D4,OUTPUT);
  digitalWrite(D4,HIGH);
  Serial.begin(9600);
  SD.begin();
  SD.remove("/dat1.csv");
  Wire.begin();
  RTC.begin();
  if (! RTC.isrunning()) {
    Serial.println("RTC is NOT running!");
    RTC.adjust(DateTime(__DATE__, __TIME__));
  }
}

void loop() {
  if (digitalRead(D3)==true){
  File f = SD.open("/dat1.csv",FILE_WRITE);
    for(int i=1; i<=30; i++){
      ara = millis();
      now = RTC.now();
      if(dht12.get()==0){
        f.print(ara/1000);
        f.print(",");
        f.print(now.day());
        f.print("/");
        f.print(now.month());
        f.print("/");
        f.print(now.year()%2000);
        f.print(" ");
        f.print(now.hour());
        f.print(":");
        f.print(now.minute());
        f.print(":");
        f.print(now.second());
        f.print(",");
        f.println(dht12.cTemp);
        espera(1000);
      }
    }
```

```
  f.close();
  }
  else {
    digitalWrite(D4,LOW);
    WiFi.begin ( ssid, password );
    while ( WiFi.status() != WL_CONNECTED ) {
      delay ( 500 );
    }
    Serial.println ( "" );
    Serial.print ( "Connected to " );
    Serial.println ( ssid );
    Serial.print ( "IP address: " );
    Serial.println ( WiFi.localIP() );
    server.on("/dat1.csv", HTTP_GET, sendFile);
    server.begin();
    while(true){
      server.handleClient();
    }
  }
}
```

Ambient light Shield (BH1750)

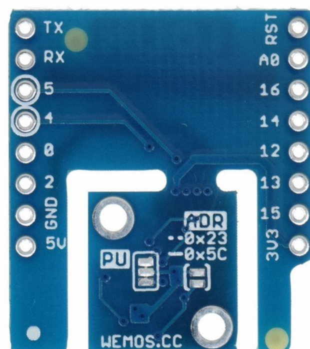


Aquest shield és equivalent al nostre Mòdul sensor de llum I2C BH1750FVI.Podem fer servir directament el codi d'aquell mòdul.

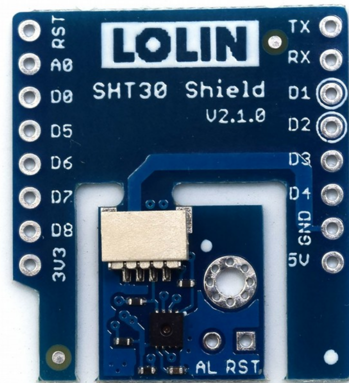
Quan es tracta de mesurar la llum cal tenir molt clar com col·loquem el sensor. El sensor en si es pot separar del shield, però ens caldria un altre shield (TFT I2C Connector Shield) per connectar-lo, ja que el connector que porta es especial. Tan especial que encara no he trobat un conjunt de connectors en bus per utilitzar una única sortida per diversos dispositius I2C.

M'estimo més fer servir el nostre mòdul: connexió a la triple base o al shield CON2 amb cables Dupont estàndard , ja està separat i en puc fer un bus de connexions I2C amb una placa de pistes i unes tires de pins.

De totes formes és interessant conèixer l'existència d'aquest shield, per si us va bé per algun projecte.



SHT30 Shield

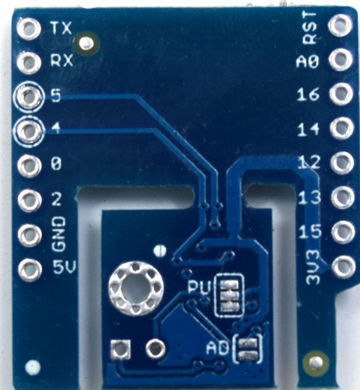


Aquest shield és semblant al nostre DHT shield, però amb mesures més precises: ± 3 % per la humitat relativa i ± 0,3 °C i amb una resolució de ± 0,05 °C amb un rang entre - 40 °C i + 125 °C. També funciona per I2C, i porta una sortida d'alarma programable.

El seu ús és tan senzill com el del DHT gracies a la llibreria *WEMOS_SHT3x*[19].

De fet, és una bona alternativa al DHT pel kit, i més potent.

A l'exemple *SHT30_OLED* mostrem al display la temperatura i la humitat cada segon:




```
#include <Wire.h>
#include <WEMOS_SHT3X.h>
#include <SFE_MicroOLED.h>  // Include the SFE_MicroOLED library
#define PIN_RESET 255  // Connect RST to pin 9
#define DC_JUMPER 0

MicroOLED oled(PIN_RESET, DC_JUMPER);    // I2C declaration
SHT3X sht30(0x45);

void setup() {
  oled.begin();    // Initialize the OLED
```

19 Disponible a https://github.com/wemos/WEMOS_SHT3x_Arduino_Library

```
  oled.clear(ALL); // Clear the display's internal memory
  oled.display();  // Display what's in the buffer (splashscreen)
  delay(1000);     // Delay 1000 ms
  oled.clear(PAGE); // Clear the buffer.

  Serial.begin(115200);
}

void loop() {
  sht30.get();
  oled.clear(PAGE);
  oled.setFontType(1);
  oled.setCursor(0,0);
  oled.print("T:");
  oled.print(sht30.cTemp);
  oled.setCursor(0,24);
  oled.print("H:");
  oled.print(sht30.humidity);
  oled.display();
  delay(1000);

}
```

Barometric Pressure Shield



Aquest shield I2C porta un sensor de pressió i temperatura HP303B.

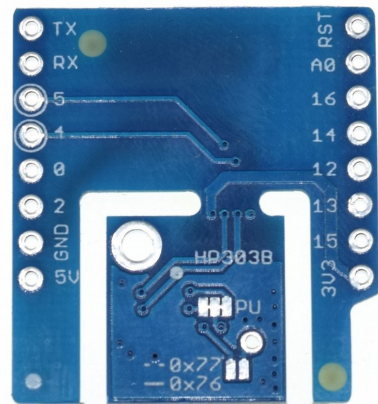
Pressió

Rang: 300 : 1200 hPa
Precisió: ± 0,005 hPa
Exactitud: ± 1 hPa

Temperatura

Rang: -40 °C : +85 °C
Exactitud: ± 0,5 °C

Com veieu, una sonda perfecta per a una estació meteorològica de butxaca.



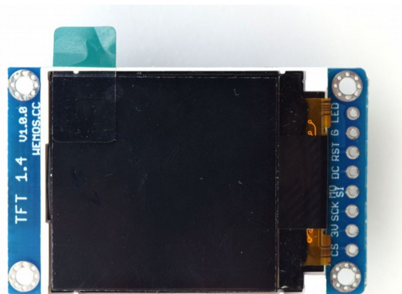
Veiem l'exemple *barometric_test*:

```
#include <LOLIN_HP303B.h>
LOLIN_HP303B HP303BPressureSensor;

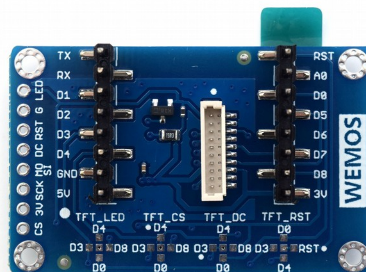
void setup()
{
  Serial.begin(9600);
  HP303BPressureSensor.begin();
  Serial.println("Barometric test");
}
```

```
void loop()
{
  int32_t temperature;
  int32_t pressure;
  int16_t oversampling = 7;
  int16_t ret;
  ret = HP303BPressureSensor.measureTempOnce(temperature, oversampling);
  Serial.print("T=");
  Serial.print(temperature);
  Serial.print(" °C, ");
  ret = HP303BPressureSensor.measurePressureOnce(pressure, oversampling);
  Serial.print("P=");
  Serial.print(pressure);
  Serial.println(" Pa");
  delay(500);
}
```

TFT 1.4 Shield



Aquest shield porta una pantalla TFT color de 128x128 px, 1,4". Utilitza un protocol SPI, amb els pins D3, D4, D5 i D7 per defecte. També afecta al pin D6.
Amb els seus 262,144 colors aquesta pantalla és ideal per mostrar gràfiques, imatges ... I al mateix preu que el nostre display OLED monocrom!
És més ampla que una columna, amb la qual cosa haurem de veure com distribuir els shields per alçades si utilitzem una base. Aquest ha estat el motiu, juntament amb el nombre de pins utilitzats, pel que vaig escollir l’OLED pel nostre kit.



Es pot connectar de 3 formes diferents:

- a una columna d’una base
- amb una tira de pins
- amb un connector especific TFT al TFT I2C Connector Shield

Utilitza les llibreries *Adafruit_GFX* i *Adafruit_ST7735*, que podeu instal·lar amb el gestor.

Com veieu a la imatge alguns pins es poden canviar. Fins i tot controlar més coses (RST independent, LED), incrementant la despesa de pins.

Cal dir que els pins D5 (SCK), D6 (MISO) i D7 (MOSI) formen la part fixa del bus SPI, i per tant es poden compartir amb altres dispositius SPI, com ara el datalogger shield. Amb D4 (TFT_CS) el bus accedeix a la pantalla TFT, i amb D8 (CS) a la SD. En canvi, com el microSD shield també fa servir D4 (TF-CS) per accedir a la SD, caldria canviar el CS per defecte d'un dels dos shields per utilitzar-los a la vegada.

L'exemple *TFT_1.4_Bitmap* , adaptat de la web d'Adafruit[20] pel nostre shield, carrega una imatge de la SD virtual i la mostra en pantalla:




```
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library
#include <SPI.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>

#define TFT_RST -1 //for TFT I2C Connector Shield V1.0.0 and TFT 1.4 Shield V1.0.0
#define TFT_CS D4  //for TFT I2C Connector Shield V1.0.0 and TFT 1.4 Shield V1.0.0
#define TFT_DC D3  //for TFT I2C Connector Shield V1.0.0 and TFT 1.4 Shield V1.0.0

// #define TFT_RST -1   //for TFT I2C Connector Shield (V1.1.0 or later)
// #define TFT_CS D0    //for TFT I2C Connector Shield (V1.1.0 or later)
// #define TFT_DC D8    //for TFT I2C Connector Shield (V1.1.0 or later)

Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

void setup(void)
```

20 Veure https://learn.adafruit.com/adafruit-1-44-color-tft-with-micro-sd-socket/drawing-bitmaps

```
{
  SPIFFS.begin();
  tft.initR(INITR_144GREENTAB);
  tft.setTextWrap(false); // Allow text to run off right edge
  tft.fillScreen(ST7735_BLACK);
  Serial.begin(9600);
}

void loop(void)
{
  tft.fillScreen(ST7735_BLACK);
  // change the name here!
  bmpDraw("lily128.bmp", 0, 0);
  // wait 5 seconds
  delay(5000);
}

// This function opens a Windows Bitmap (BMP) file and
// displays it at the given coordinates.  It's sped up
// by reading many pixels worth of data at a time
// (rather than pixel by pixel).  Increasing the buffer
// size takes more of the Arduino's precious RAM but
// makes loading a little faster.  20 pixels seems a
// good balance.
#define BUFFPIXEL 20

void bmpDraw(char *filename, uint8_t x, uint16_t y) {
  File     bmpFile;
  int      bmpWidth, bmpHeight;   // W+H in pixels
  uint8_t  bmpDepth;              // Bit depth (currently must be 24)
  uint32_t bmpImageoffset;        // Start of image data in file
  uint32_t rowSize;               // Not always = bmpWidth; may have padding
  uint8_t  sdbuffer[3*BUFFPIXEL]; // pixel buffer (R+G+B per pixel)
  uint8_t  buffidx = sizeof(sdbuffer); // Current position in sdbuffer
  boolean  goodBmp = false;       // Set to true on valid header parse
  boolean  flip    = true;        // BMP is stored bottom-to-top
  int      w, h, row, col;
  uint8_t  r, g, b;
  uint32_t pos = 0, startTime = millis();
  if((x >= tft.width()) || (y >= tft.height())) return;
  Serial.println();
  Serial.print(F("Loading image '"));
  Serial.print(filename);
  Serial.println('\'');
  // Open requested file on SD card
  if ((bmpFile = SPIFFS.open("/lily128.bmp","r")) == NULL) {
    Serial.print(F("File not found"));
    return;
  }
  // Parse BMP header
  if(read16(bmpFile) == 0x4D42) { // BMP signature
    Serial.print(F("File size: ")); Serial.println(read32(bmpFile));
    (void)read32(bmpFile); // Read & ignore creator bytes
    bmpImageoffset = read32(bmpFile); // Start of image data
    Serial.print(F("Image Offset: ")); Serial.println(bmpImageoffset, DEC);
    // Read DIB header
    Serial.print(F("Header size: ")); Serial.println(read32(bmpFile));
    bmpWidth  = read32(bmpFile);
    bmpHeight = read32(bmpFile);
    if(read16(bmpFile) == 1) { // # planes -- must be '1'
      bmpDepth = read16(bmpFile); // bits per pixel
      Serial.print(F("Bit Depth: ")); Serial.println(bmpDepth);
      if((bmpDepth == 24) && (read32(bmpFile) == 0)) { // 0 = uncompressed
        goodBmp = true; // Supported BMP format -- proceed!
```

```
        Serial.print(F("Image size: "));
        Serial.print(bmpWidth);
        Serial.print('x');
        Serial.println(bmpHeight);
        // BMP rows are padded (if needed) to 4-byte boundary
        rowSize = (bmpWidth * 3 + 3) & ~3;
        // If bmpHeight is negative, image is in top-down order.
        // This is not canon but has been observed in the wild.
        if(bmpHeight < 0) {
          bmpHeight = -bmpHeight;
          flip      = false;
        }
        // Crop area to be loaded
        w = bmpWidth;
        h = bmpHeight;
        if((x+w-1) >= tft.width())  w = tft.width()  - x;
        if((y+h-1) >= tft.height()) h = tft.height() - y;
        // Set TFT address window to clipped image bounds
        tft.startWrite();
        tft.setAddrWindow(x, y, w, h);
        for (row=0; row<h; row++) { // For each scanline...
          // Seek to start of scan line.  It might seem labor-
          // intensive to be doing this on every line, but this
          // method covers a lot of gritty details like cropping
          // and scanline padding.  Also, the seek only takes
          // place if the file position actually needs to change
          // (avoids a lot of cluster math in SD library).
          if(flip) // Bitmap is stored bottom-to-top order (normal BMP)
            pos = bmpImageoffset + (bmpHeight - 1 - row) * rowSize;
          else     // Bitmap is stored top-to-bottom
            pos = bmpImageoffset + row * rowSize;
          if(bmpFile.position() != pos) { // Need seek?
            tft.endWrite();
            bmpFile.seek(pos);
            buffidx = sizeof(sdbuffer); // Force buffer reload
          }
          for (col=0; col<w; col++) { // For each pixel...
            // Time to read more pixel data?
            if (buffidx >= sizeof(sdbuffer)) { // Indeed
              bmpFile.read(sdbuffer, sizeof(sdbuffer));
              buffidx = 0; // Set index to beginning
              tft.startWrite();
            }

            // Convert pixel from BMP to TFT format, push to display
            b = sdbuffer[buffidx++];
            g = sdbuffer[buffidx++];
            r = sdbuffer[buffidx++];
            tft.pushColor(tft.color565(r,g,b));
          } // end pixel
        } // end scanline
        tft.endWrite();
        Serial.print(F("Loaded in "));
        Serial.print(millis() - startTime);
        Serial.println(" ms");
      } // end goodBmp
    }
  }
  bmpFile.close();
  if(!goodBmp) Serial.println(F("BMP format not recognized."));
}

// These read 16- and 32-bit types from the SD card file.
// BMP data is stored little-endian, Arduino is little-endian too.
```

```
// May need to reverse subscript order if porting elsewhere.

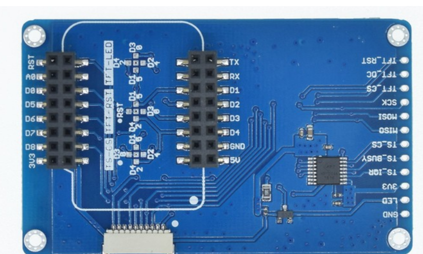
uint16_t read16(File f) {
  uint16_t result;
  ((uint8_t *)&result)[0] = f.read(); // LSB
  ((uint8_t *)&result)[1] = f.read(); // MSB
  return result;
}

uint32_t read32(File f) {
  uint32_t result;
  ((uint8_t *)&result)[0] = f.read(); // LSB
  ((uint8_t *)&result)[1] = f.read();
  ((uint8_t *)&result)[2] = f.read();
  ((uint8_t *)&result)[3] = f.read(); // MSB
  return result;
}
```

TFT 2.4 Touch Shield

Aquest shield ofereix una pantalla 320x240 TFT de 2,4" sensible al tacte.
Utilitza les biblioteques *Adafruit_GFX*, *Adafruit_ILI9341* i *XPT2046_Touchscreen.*
La connexió del bus de columna es femella. També es pot connectar amb una tira de pins o amb el TFT I2C Connector Shield.
Utilitza els pins D0, D3, D5, D6, D7 i D8, amb un bus SPI.
Especialment interessant per a menús interactius, nsimulacions de joystick, signatures ...





L'exemple *TFT2p4_touchpaint*, adaptat de la web d'Adafruit[21] per a aquest shield, permet dibuixar en la pantalla amb el dit:



21 Veure https://learn.adafruit.com/adafruit-2-4-tft-touch-screen-featherwing/resistive-touch-screen

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <XPT2046_Touchscreen.h>

#define TFT_CS D0  //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)
#define TFT_DC D8  //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)
#define TFT_RST -1 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)
#define TS_CS D3   //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);
XPT2046_Touchscreen ts(TS_CS);

// This is calibration data for the raw touch data to the screen coordinates
#define TS_MINX 3800
#define TS_MAXX 100
#define TS_MINY 100
#define TS_MAXY 3750

// Size of the color selection boxes and the paintbrush size
#define BOXSIZE 40
#define PENRADIUS 3
int oldcolor, currentcolor;

void setup(void) {
  Serial.begin(115200);
  delay(10);
  Serial.println("FeatherWing TFT");
  if (!ts.begin()) {
    Serial.println("Couldn't start touchscreen controller");
    while (1);
  }
  ts.setRotation(4);
  Serial.println("Touchscreen started");
  tft.begin();
  tft.setRotation(2);
  tft.fillScreen(ILI9341_BLACK);

  // make the color selection boxes
  tft.fillRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_RED);
  tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_YELLOW);
  tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_GREEN);
  tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_CYAN);
  tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_BLUE);
  tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_MAGENTA);

  // select the current color 'red'
  tft.drawRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  currentcolor = ILI9341_RED;
}

void loop() {
  // Retrieve a point
  TS_Point p = ts.getPoint();
  Serial.print("X = "); Serial.print(p.x);
  Serial.print("\tY = "); Serial.print(p.y);
  Serial.print("\tPressure = "); Serial.println(p.z);

  // Scale from ~0->4000 to tft.width using the calibration #'s
  p.x = map(p.x, TS_MINX, TS_MAXX, 0, tft.width());
  p.y = map(p.y, TS_MINY, TS_MAXY, 0, tft.height());

  if (p.y < BOXSIZE) {
     oldcolor = currentcolor;
```

```
      if (p.x < BOXSIZE) {
        currentcolor = ILI9341_RED;
        tft.drawRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
      } else if (p.x < BOXSIZE*2) {
        currentcolor = ILI9341_YELLOW;
        tft.drawRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
      } else if (p.x < BOXSIZE*3) {
        currentcolor = ILI9341_GREEN;
        tft.drawRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
      } else if (p.x < BOXSIZE*4) {
        currentcolor = ILI9341_CYAN;
        tft.drawRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
      } else if (p.x < BOXSIZE*5) {
        currentcolor = ILI9341_BLUE;
        tft.drawRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
      } else if (p.x < BOXSIZE*6) {
        currentcolor = ILI9341_MAGENTA;
        tft.drawRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
      }

      if (oldcolor != currentcolor) {
         if (oldcolor == ILI9341_RED)
           tft.fillRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_RED);
         if (oldcolor == ILI9341_YELLOW)
           tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_YELLOW);
         if (oldcolor == ILI9341_GREEN)
           tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_GREEN);
         if (oldcolor == ILI9341_CYAN)
           tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_CYAN);
         if (oldcolor == ILI9341_BLUE)
           tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_BLUE);
         if (oldcolor == ILI9341_MAGENTA)
           tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_MAGENTA);
      }
   }

   if (((p.y-PENRADIUS) > 0) && ((p.y+PENRADIUS) < tft.height())) {
     tft.fillCircle(p.x, p.y, PENRADIUS, currentcolor);
   }
}
```

TFT I2C Connector Shield

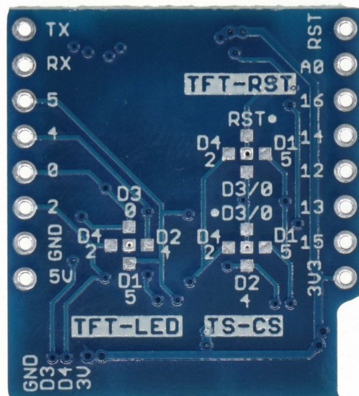


Aquest shield porta uns connectors especials pels shields TFT i pels sensors I2C.

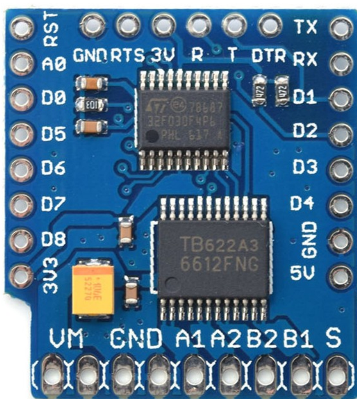
Especialment útil quan el display TFT el volem separar de la base del D1 mini.

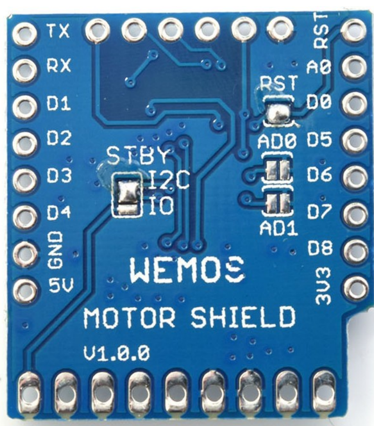
Aneu amb compte amb la versió del connector. Han canviats coses. La V1.0.0, pensada amb el TFT 1.4 Shield, portava un selector per TFT_CS, per defecte connectat a D4. La versió actual V1.1.0, pensada amb el TFT 2.4 Touch Shield, no porta aquest selector, TFT_CS va connectat ara sempre a D0. El mateix passa amb TFT_DC. Abans portava un selector, per defecte a D3. Ara va fixe a D8.

A més a més la V1.1.0 porta un nou connector I/O amb D3 i D4. Aquesta darrera connexió amb D3 no té sentit si no s'utilitza el TFT 1.4 Shield, que no té sensor tàctil.



Motor shield





Aquest shield porta un pont H controlat pel seu propi microcontroladors al que s'accedeix amb I2C.

D'aquesta manera podem controlar 2 motors fàcilment. Podem aturar-los o engegar-los, canviar la velocitat, el sentit de gir, frenar-los o deixar-los en stand-by de forma independent

Utilitza la llibreria pròpia WEMOS_Motor_Shield[22], i el firmware[23] es pot modificar.

Pot donar una intensitat promig de 1,2 A i 3,2 A de pic, amb una alimentació independent pels motors de fins a 15 V.

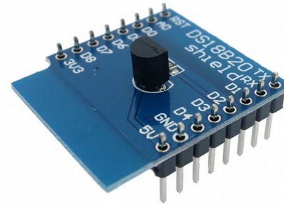
Personalment, per a aplicacions de tipus robots m'estimo més utilitzar Els mòduls ESP12 V2. La placa NODEMCU motor shield . Aquesta altra placa porta interruptor i regulador de tensió pel mòdul ESP8266, així com connectors per a sensors externs.

22 Disponible a https://github.com/wemos/WEMOS_Motor_Shield_Arduino_Library

23 Disponible a https://github.com/wemos/Motor_Shield_Firmware

DS18B20 shield





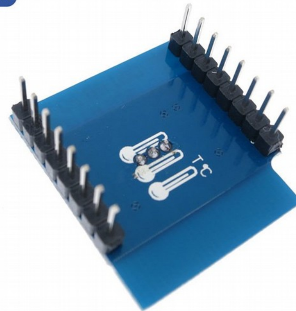
Aquest shield porta un sensor de temperatura DS18B20, essent equivalent al nostre Mòdul sensor temperatura DS18B20 .

El problema és que va connectat al pin D2, inutilitzant el bus I2C. A més a més, no porta connector per altres sensors al mateix bus OneWire.

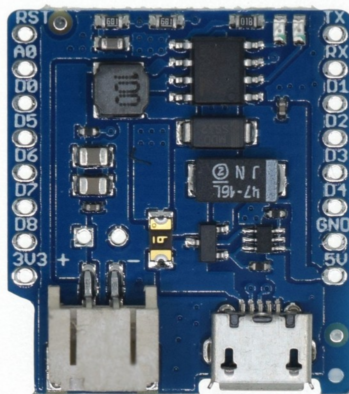
Pot ser interessant quan no fem servir el bus I2C, o per wearables on podem canviar la connexió amb el fil.

Podeu fer servir els mateixos programes que tenim pel nostre mòdul. Això si, cal canviar el pin per D2.





Battery Shield



Es tracta d’un shield al qual podem connectar una bateria de Liti (3,3 V : 4,2 V) amb un connector PH2-2.0MM o bé soldada a la placa.

La bateria es carrega amb el seu propi connector microUSB. Podem configurar el shield per carregar amb un màxim de 0,5 A (per defecte) o 1 A.

El shield genera 5 V / 1 A amb la bateria i els injecta a l’alimentació del bus del D1 mini.

Porta dos leds: vermell mentre carrega i verd quan la càrrega és completa.

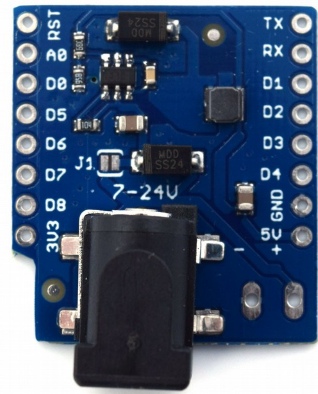
Podem configurar el shield per connectar l‘entrada analògica A0 a la bateria i així monitoritzar el seu estat.



DC Power Shield

Aquest shield porta un regulador de tensió de 5 V / 1 A que injecta al bus D1 mini.

Podem alimentar el sistema amb una entrada de 7 V : 24 V connectats amb un connector de barril, soldant a la placa o amb una regleta





Part VI. HTML5, javaScript i AJAX

El llenguatge HTML va ser dissenyat al CERN per Tim Berners-Lee i Robert Caillau[W3C01] al 1991 per interconnectar estacions de treball gràfiques X-Window. Per compartir informació entre ordinadors diferents.

En la evolució de la web, empreses amb programari propietari han corromput aquesta filosofia: extensions del navegador de Microsoft, plugins flash i java ... han dificultat crear continguts amb aquest estàndard multiplataforma.

HTML5 torna a posar les coses al seu lloc. Amb la seva potència no hi ha excusa per altres extensions o plugins sinó és per ànsia de poder de certes companyies.

HTML5 va estretament lligat a javascript i CSS. El primer per fer codi que s'executa al navegador del client, estalviant recursos al servidor i transit de dades a la xarxa, i accelerant la resposta de la pàgina. Objectius prioritaris quan el nostre servidor és un modest ESP8266, que, a més a més, és possible que hagi generat la seva pròpia xarxa.

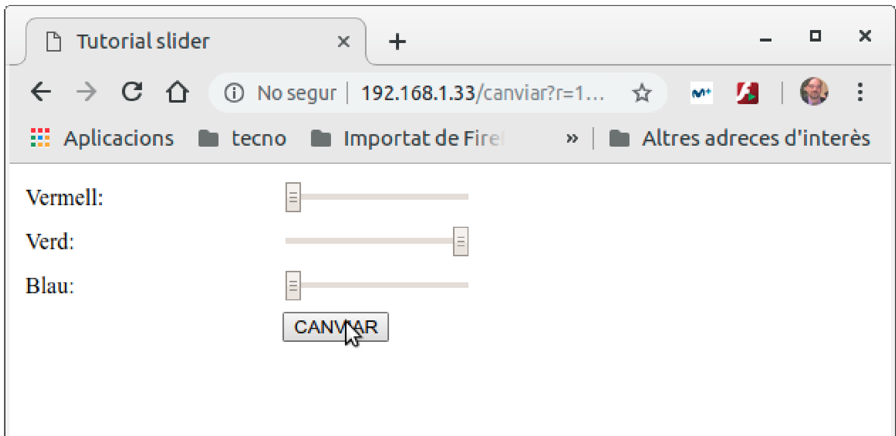
Al llarg d'aquest capítol veurem alguns dels recursos que HTML5, javascript i CSS ens ofereixen per millorar les nostres interfícies de xarxa. Però recordeu que això és només un tasteig, les possibilitats[24] són il·limitades!

24 Us recomano la fantàstica web https://www.w3schools.com/. Els seus tutorials, referències i exemples són genials!

HTML5 input type='range'. Creació de sliders.

HTML5 introdueix nous tipus d'entrades als formularis[25]. Comencem amb el tipus *range*, que genera una barra de selecció o slider.

Veiem un exemple (*slider*) on fem servir tres sliders per modificar les components de color del led central del nostre shield RGB:




```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char *ssid = "Aquí el nom de la teva xarxa";
const char *password = "Aquí la contrasenya de la xarxa";

ESP8266WebServer server ( 80 );

#include <Adafruit_NeoPixel.h>
#define PIN            D8
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ800);

String mensaje = "";

//-----------CODIGO HTML PAGINA DE CONFIGURACION---------------
String pagina = "<!DOCTYPE html>"
"<html>"
"<head>"
"<title>Tutorial slider</title>"
"<meta charset='UTF-8'>"
"<meta name='viewport' content='width=device-width'/>"
"</head>"
```

25 Veure https://www.w3schools.com/html/html_form_input_types.asp

```
"<body>"
"<table style='width:100%'><form action='canviar' method='get'>"
"<tr><td>Vermell:</td><td><input type='range' name='r' min='0'
max='255'></td></tr>"
"<tr><td>Verd:</td><td><input type='range' name='g' min='0'
max='255'></td></tr>"
"<tr><td>Blau:</td><td><input type='range' name='b' min='0'
max='255'></td></tr>"
"<tr><td></td><td><input type='submit' value='CANVIAR' /></td></tr>"
"</form></table>"
"</body>"
"</html>";

void paginacanvi() {
  server.send(200, "text/html", pagina);
}

void canviar_colors() {
  pixels.setPixelColor(0, server.arg("r").toInt(), server.arg("g").toInt(),
server.arg("b").toInt());
  pixels.show(); // This sends the updated pixel color to the hardware.
  paginacanvi();
}

void setup() {
  WiFi.begin ( ssid, password );
  pixels.begin();
  server.on("/", paginacanvi);
  server.on("/canviar", canviar_colors);
  server.begin();

  pixels.setPixelColor(0, pixels.Color(0, 63, 0));
  pixels.show();

}

void loop() {
  server.handleClient();
}
```

Primer de tot fixeu-vos que he canviat el D2 original per D8 a la línia

```
#define PIN            D8
```

ja que aquest exemple inicialment va ser fet pel WS2812B RGB Shield, inclòs a la primera versió del kit.

Us recomano afegir sempre al *<head>* la línia

```
"<meta name='viewport' content='width=device-width'/>"
```

que adapta la visualització de la pàgina a l’amplada del dispositiu. Imprescindible per mostrar correctament la pàgina a dispositius mòbils.

Fixeu-vos que al codi HTML sempre faig servir la cometa simple ('). Això permet incloure el codi HTML al codi arduino amb cometes dobles (").

Per crear al formulari el slider fem servir el codi

```
<input type='range' name='g' min='0' max='255'>
```

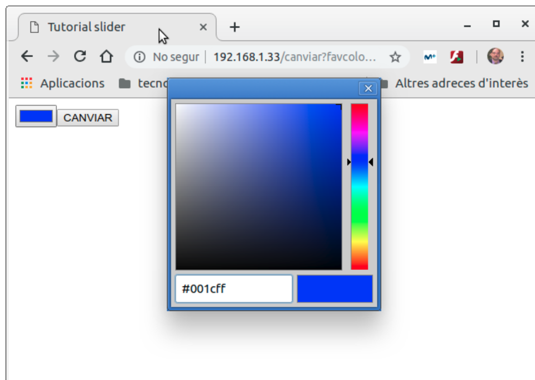
Aquest valor el recuperem i interpretem com a nombre amb el codi

```
server.arg("g").toInt()
```

Això ho fem pels tres colors RGB, cadascú amb el seu propi paràmetre.

HTML5 input type='color'

A l'exemple *Color_Picker* seleccionem el color del nostre RGB shield amb un selector de color. La seva aparença canvia segons el dispositiu:






```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char *ssid = "Aquí el nom de la teva xarxa";
const char *password = "Aquí la contrasenya de la xarxa";

ESP8266WebServer server ( 80 );
#include <Adafruit_NeoPixel.h>
#define PIN            D8
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ800);
```

```
String mensaje = "";

//-----------CODIGO HTML PAGINA DE CONFIGURACION---------------
String pagina = "<!DOCTYPE html>"
"<html>"
"<head>"
"<title>Tutorial slider</title>"
"<meta charset='UTF-8'>"
"<meta name='viewport' content='width=device-width'/>"
"</head>"
"<body>"
"<form action='canviar' method='get'>"
"<input type='color' name='favcolor' value='#ff0000'>"
"<input type='submit' value='CANVIAR' /></br>"
"</form>"
"</body>"
"</html>";

void paginacanvi() {
  server.send(200, "text/html", pagina);
}

int hex2dec(byte c) {
  if (c >= '0' && c <= '9') {
    return c - '0';
  } else if (c >= 'A' && c <= 'F') {
    return c - 'A' + 10;
  } else if (c >= 'a' && c <= 'f') {
    return c - 'a' + 10;
  }
}

void canviar_colors() {
  int r,g,b;
  String c;
  c = server.arg("favcolor");
  Serial.print("c= ");
  Serial.println(c);
  r=hex2dec(c.charAt(1))*16+hex2dec(c.charAt(2));
  g=hex2dec(c.charAt(3))*16+hex2dec(c.charAt(4));
  b=hex2dec(c.charAt(5))*16+hex2dec(c.charAt(6));
  Serial.print("r= ");
  Serial.println(r);
  Serial.print("g= ");
  Serial.println(g);
  Serial.print("b= ");
  Serial.println(b);
  pixels.setPixelColor(0, r, g, b);
  pixels.show(); // This sends the updated pixel color to the hardware.
  paginacanvi();
}

void setup() {
  Serial.begin(115200);
  WiFi.begin ( ssid, password );
  pixels.begin(); // This initializes the NeoPixel library.
  server.on("/", paginacanvi);
  server.on("/canviar", canviar_colors);
  server.begin();

  pixels.setPixelColor(0, pixels.Color(0, 63, 0));
  pixels.show(); // This sends the updated pixel color to the hardware.
}
```

```
void loop() {
  server.handleClient();
}
```

Amb el codi HTML5

```
<input type='color' name='favcolor' value='#ff0000'>
```

seleccionem un color en format RGB. Separem les components amb codi arduino del tipus

```
r=hex2dec(c.charAt(1))*16+hex2dec(c.charAt(2));
```

per poder cridar a la funció *setPixelColor()* amb les tres components de color separades.

Incloure eines HTML5 a la SD virtual

A la comunitat del programari lliure hi ha una quantitat immensa de programadors que han creat eines HTML5 fantàstiques que es poden aprofitar desant uns quants fitxers a la nostra SD virtual. Podem fer servir FSManager, la navalla suïssa per a la gestió de fitxers a la SD virtual via WiFi per desar aquests fitxers i accedir-hi.

A l'exemple *carrusel* he modificat l'exemple *bitmapSPIFFS* afegint-hi la funcionalitat de *FSManager* i fent un escanejat de la SD virtual per trobar arxius amb l'extensió *.dat* per mostrar els bitmaps que contenen. Per simplificar el seu ús, la funcionalitat del *FSManager* l'he posada en dos fitxers (F*SManager.cpp* i *FSManager.h*) a la mateixa carpeta del programa[26].

I, el més important, he afegit a la carpeta *data* l'eina *image2cpp*[27] que permet codificar nous bitmaps per afegir-ne.

Aquí teniu el contingut de *carrusel.ino*

```
#include <Wire.h>
#include <SFE_MicroOLED.h>
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;

#define PIN_RESET 255  //
#define DC_JUMPER 0  // I2C Addres: 0 - 0x3C, 1 - 0x3D
MicroOLED oled(PIN_RESET, DC_JUMPER);  // I2C Example
File f;
uint8_t pixels[16*24];


void loadbitmap(String fitxer);

void setup() {
  SPIFFS.begin();
  initHelper(); //inclou connexió Wifi i server
              // Aquí podem afegir altres funcions server.on()
  server.begin();

  oled.begin();
  oled.clear(ALL);
```

26 Veure Annex 9: Afegint la funcionalitat de FSManager als nostres projectes
27 Disponible a http://javl.github.io/image2cpp/

```
  oled.display();
  delay(1000);
}

void loop() {
  Dir dir = SPIFFS.openDir("/");
  while (dir.next()) {
    String fileName = dir.fileName();
    if(fileName.endsWith(".dat")){
      oled.clear(PAGE);
      loadbitmap(fileName);
      oled.drawBitmap(pixels);
      oled.display();
      espera(1000);
    }
  }

}

String getLine() {
  String S = "" ;
  char c = f.read();
  while ( c != '\n') {
    S = S + c ;
    c = f.read();
    }
  return(S) ;
}

int convertFromHex(int ascii){
  if(ascii > 0x39) ascii -= 7; // adjust for hex letters upper or lower case
  return(ascii & 0xf);
}

void loadbitmap(String fitxer){
  uint8_t sa[16];
  String txt;  int i,j,k;
  f = SPIFFS.open(fitxer,"r");
    for (i=0;i<24;i++){
      txt=getLine();
      k=0;
      for (j=0; j<16; j++) {
        while (txt.charAt(k) != 'x') k++;
        sa[j] = convertFromHex(txt[k+1])*16;
        sa[j] += convertFromHex(txt[k+2]);
        k++;
      }
      for (j=0;j<16;j++) {
          pixels[i*16+j]=sa[j];
      }
    }
  f.close();
}
```

Utilitzant AJAX

Si volem mostrar dades a la nostra interfície web de forma eficient AJAX és la solució.

Només hem de generar pàgines de text amb el nostre ESP8266 amb les dades i una bonica pàgina web amb codi javascript a la nostre servidor web. Millor fer la pàgina a la SD virtual i modificar el seu aspecte amb el nostre *FSManager* incrustat.

Veiem l'exemple *DHT_ajax*:

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;

#include <WEMOS_DHT12.h>
DHT12 dht12;

void setup() {
  SPIFFS.begin();
  pinMode(D4,OUTPUT);
  initHelper(); //inclou connexió Wifi i server
  server.on("/ajax_info.txt", HTTP_GET, []() {
    dht12.get();
    String txt = String(dht12.cTemp);
    server.send(200, "text/plain", txt);
    txt = String();
  });
  server.begin();
}

void loop() {
  digitalWrite(D4,LOW);
  espera(100);
  digitalWrite(D4,HIGH);
  espera(100);
}
```

Senzill i breu, oi?

Recordeu que a la mateixa carpeta tenim *FSManager.cpp* i *FSManager.h* que s'encarreguen de la gestió web de la SD virtual.

Ara s'entén perquè *server.begin()* no està inclòs dintre de *initHelper()*. Abans d'iniciar el servidor web cal definir les pàgines pròpies de la nostra aplicació.

Veiem el fitxer *index.htm* creat a la SD virtual:

```
<html>
<head>
  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
  <meta name='viewport' content='width=device-width'>
  <title>AJAX Test</title>
  <style>
    body {
      background-color: Cornsilk;
    }
  </style>
  <script>
    function loadTemp() {
      var xhttp = new XMLHttpRequest();
      var temp = 0;
      xhttp.open('GET', 'ajax_info.txt', false);
      xhttp.send();
      temp = parseFloat(xhttp.responseText);
      return (temp);
    }
  </script>
  <style>
    #valor {
                 border-radius: 25px;
                 background: #73AD21;
                 padding: 20px;
                 width: 55px;
                 height: 12px;
                 text-align: center;
    }
  </style>
</head>
<body>
<h1>Temperatura</h1>
<div style="padding-left:40px;padding-bottom:5px;">
<p id='valor'></p>
</div>
<p><img id='icona' src='cloud.svg'></p>
<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
  var t= loadTemp();
  document.getElementById('valor').innerHTML = t+' °C';
  if (t>27) {
    document.getElementById('icona').src = 'sun.svg';
  }
  else {
    document.getElementById('icona').src = 'snowflake.svg';
  }
}
</script>
</body>
</html>
```

Tota la màgia està en la funció myTimer(). No només actualitza el valor de la temperatura amb

```
  var t= loadTemp();
  document.getElementById('valor').innerHTML = t+' °C';
```

A més a més canvia la imatge de sota en funció del valor de la temperatura:

```
  if (t>27) {
    document.getElementById('icona').src = 'sun.svg';
  }
  else {
    document.getElementById('icona').src = 'snowflake.svg';
  }
```

El truc és utilitzar *id* al codi HTML:

```
<p id='valor'></p>
```

que és el mateix *id* que fem servir per canviar el contingut en aquell paràgraf:

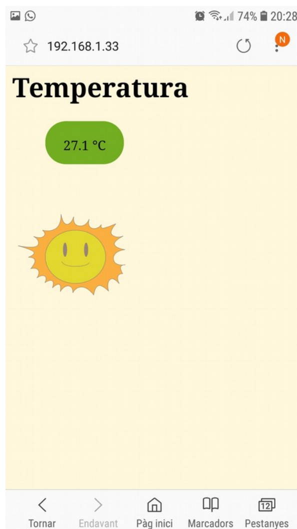
```
  document.getElementById('valor').innerHTML = t+' °C';
```

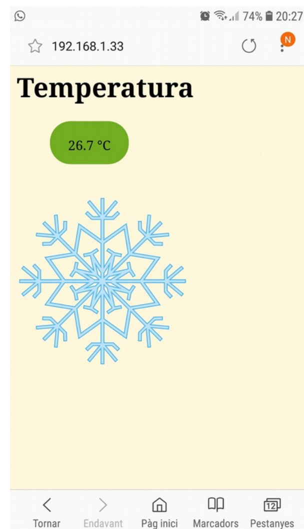
Amb un altre *id* identifiquem la imatge mostrada:

```
<p><img id='icona' src='cloud.svg'></p>
```

i la modifiquem:

```
    document.getElementById('icona').src = 'sun.svg';
```





Naturalment a la nostra SD virtual tenim les tres imatges SVG:



I tot això amb només 48 kB de la nostra SD virtual!

Fixeu-vos que poc ocupen les imatges. Són gràfics vectorials, re-escalables i molt eficients.

Fixeu-vos també com el codi CSS millora l'aspecte de la interfície. Aquest codi dona l'aparença d'un botó arrodonit a la temperatura:

```
<style>
  #valor {
              border-radius: 25px;
              background: #73AD21;
              padding: 20px;
              width: 55px;
              height: 12px;
              text-align: center;
  }
</style>
```

Fixeu-vos que valor és l'*id* del paràgraf on posem la temperatura, i va precedit pel signe sostingut (#).

La potència de javascript. Gràfics HTML canvas

Javascript és un gran llenguatge, que se executa al navegador del nostre ordinador o mòbil, amb una potència de càlcul molt superior al nostre modest ESP8266. Cal aprofitar aquest avantatge a les nostres interfícies HTML.

HTML5 permet dos tipus de gràfics. SVG i canvas. SVG és vectorial, canvas orientat a bitmap. Cadascú té els seus avantatges i desavantatges.

En el meu cas m'interessa treballar amb canvas, ja que permet baixar directament la imatge com un arxiu png, i SVG no.

Fa poc em vaig fer una càmera tèrmica basada en el nostre kit, de forma que porta FSManager incrustat en el codi per servir els fitxers de les imatges capturades, que consisteixen en una matriu 8x8 de valors de la temperatura. Vull baixar a l'ordinador les imatges amb els colors mapejats tèrmicament per fer un tractament digital de la imatge i utilitzar-les en altres treballs.

Veiem l'exemple *visor_foto_term*:

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;

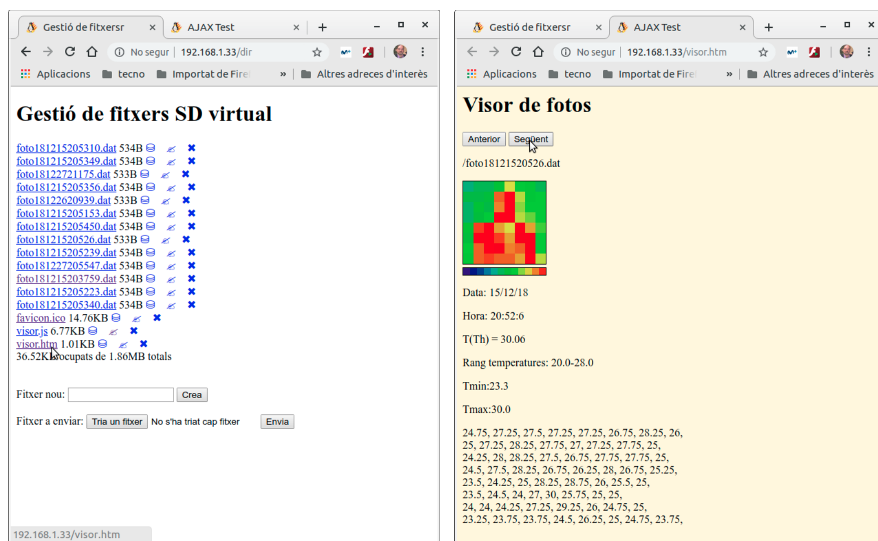
void setup() {
  SPIFFS.begin();
  pinMode(D4,OUTPUT);
  initHelper(); //inclou connexió Wifi i server
  server.begin();
}

void loop() {
  digitalWrite(D4,LOW);
  espera(100);
  digitalWrite(D4,HIGH);
  espera(100);
}
```

Si. Només fa un blink. O això sembla ...

A veure. En realitat el que m’interessa es posar a les vostres mans un sistema que funcioni i us permeti seguir l’exemple. Un ESP8266 amb el FSManager incrustat i amb imatges tèrmiques reals desades a la SD virtual. La meva càmera té un codi més llarg[28], ja que ha de fer les captures d’imatge, mostrar les imatges al TFT 1.4 Shield que porta i les dades al OLED Shield ... I, naturalment, crear el servidor amb el FSManager, que és la part que treballarem, on el veritable codi és a la interfície web, concretament als arxius *visor.htm* i *visor.js*.

Veiem primer el resultat:



Quan visualitzo el fitxer *visor.htm* aconsegueixo una interfície web que em permet desplaçar-me pels diferents arxius .dat de la SD virtual i visualitzar el mapa tèrmic i les diferents dades capturades. Tot utilitzant menys de 8 kB de la SD virtual!

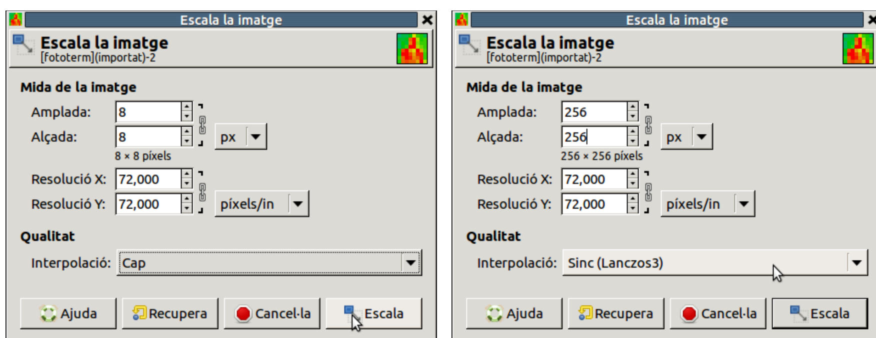
---

28 El codi de la càmera el teniu disponible a https://github.com/jorts64/kit-D1-mini/tree/master/projectes/jorts/camera%20termica

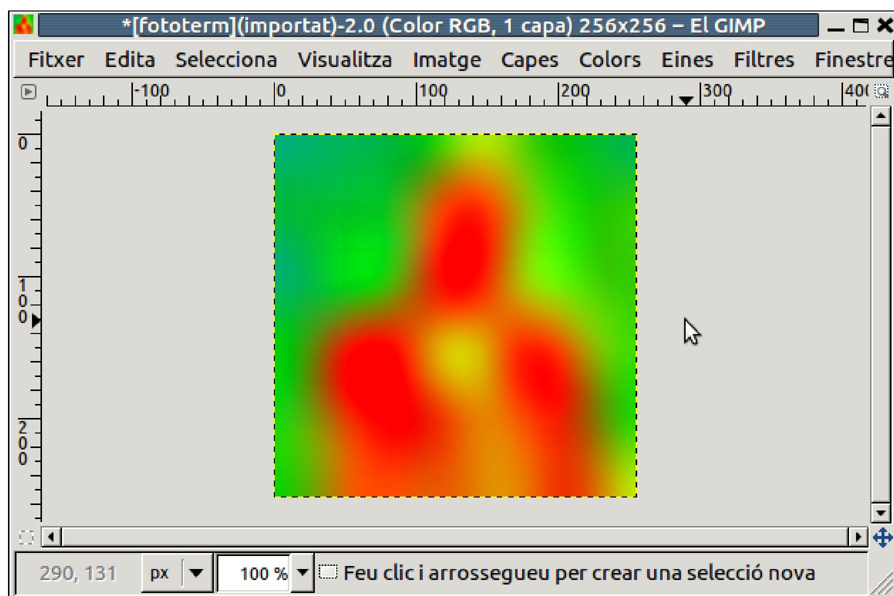
Fixeu-vos la reduïda mida dels fitxers d'imatge. A continuació teniu el contingut del fitxer .dat de la imatge mostrada:

```
15/12/18
20:52:6
T(Th) = 30.06
20.0-28.0
Tmin:23.3
Tmax:30.0
24.75, 27.25, 27.50, 27.25, 27.25, 26.75, 28.25, 26.00,
25.00, 27.25, 28.25, 27.75, 27.00, 27.25, 27.75, 25.00,
24.25, 28.00, 28.25, 27.50, 26.75, 27.75, 27.75, 25.00,
24.50, 27.50, 28.25, 26.75, 26.25, 28.00, 26.75, 25.25,
23.50, 24.25, 25.00, 28.25, 28.75, 26.00, 25.50, 25.00,
23.50, 24.50, 24.00, 27.00, 30.00, 25.75, 25.00, 25.00,
24.00, 24.00, 24.25, 27.25, 29.25, 26.00, 24.75, 25.00,
23.25, 23.75, 23.75, 24.50, 26.25, 25.00, 24.75, 23.75,
```

Clicant amb el botó dret sobre la imatge la puc baixar com a arxiu .png. Si amb el GIMP redueixo la imatge a les dades originals (8x8) i faig una interpolació *Lanczos3* per aconseguir una imatge 256x256 :



aconsegueixo una imatge millorada digitalment que difícilment podria obtenir amb l'ESP8266, i amb una transmissió de dades molt petita comparada amb el resultat:



Impressionant, oi?

Al codi HTML carreguem les funcions javascript del fitxer *visor.js* al <*head*> i definim els diferents espais gràfics <*canvas*> i textuals <*p*>, cadascú amb el seu *id*, així com el botons per poder moure'ns entre les imatges. Finalment llegim el directori amb AJAX i visualitzem la primera fotografia. Aquí teniu el contingut de *visor.htm*:

```
<html>
<head>
  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
  <meta name='viewport' content='width=device-width'>
  <title>AJAX Test</title>
  <style>
    body {
      background-color: Cornsilk;
    }
  </style>
  <script src="visor.js"></script>
</head>
<body>
<h1>Visor de fotos</h1>
<div>
<button onclick="anterior()">Anterior</button>
<button onclick="seguent()">Següent</button>
</div>
```

```
<p id='nomf'></p>
<canvas id="foto" width="120" height="120" style="border:1px solid #000000;">
</canvas>
<br/>
<canvas id="escala" width="120" height="10" style="border:1px solid #000000;">
</canvas>
<p id='crues'></p>
<p id='fecha'></p>
<p id='hora'></p>
<p id='tth'></p>
<p id='rangt'></p>
<p id='mintemp'></p>
<p id='maxtemp'></p>
<p id='tmin'></p>
<p id='tmax'></p>
<p id='test'></p>
<p id='array'></p>

<script>
var arxius = getDir();
//document.getElementById('test').innerHTML = arxius;
fotonum = 0;
var myVar = loadArray(arxius[fotonum]);
</script>
</body>
</html>
```

El fitxer HTML ha quedat elegant perquè he separat les funcions javascript a un altre fitxer *visor.js*:

```
camColors = [0x480F,
0x400F,0x400F,0x400F,0x4010,0x3810,0x3810,0x3810,0x3810,0x3010,0x3010,
0x3010,0x2810,0x2810,0x2810,0x2810,0x2010,0x2010,0x2010,0x1810,0x1810,
0x1811,0x1811,0x1011,0x1011,0x1011,0x0811,0x0811,0x0811,0x0011,0x0011,
0x0011,0x0011,0x0011,0x0031,0x0031,0x0051,0x0072,0x0072,0x0092,0x00B2,
0x00B2,0x00D2,0x00F2,0x00F2,0x0112,0x0132,0x0152,0x0152,0x0172,0x0192,
0x0192,0x01B2,0x01D2,0x01F3,0x01F3,0x0213,0x0233,0x0253,0x0253,0x0273,
0x0293,0x02B3,0x02D3,0x02D3,0x02F3,0x0313,0x0333,0x0333,0x0353,0x0373,
0x0394,0x03B4,0x03D4,0x03D4,0x03F4,0x0414,0x0434,0x0454,0x0474,0x0474,
0x0494,0x04B4,0x04D4,0x04F4,0x0514,0x0534,0x0534,0x0554,0x0554,0x0574,
0x0574,0x0573,0x0573,0x0573,0x0572,0x0572,0x0572,0x0571,0x0591,0x0591,
0x0590,0x0590,0x058F,0x058F,0x058F,0x058E,0x05AE,0x05AE,0x05AD,0x05AD,
0x05AD,0x05AC,0x05AC,0x05AB,0x05CB,0x05CB,0x05CA,0x05CA,0x05CA,0x05C9,
0x05C9,0x05C8,0x05E8,0x05E8,0x05E7,0x05E7,0x05E6,0x05E6,0x05E6,0x05E5,
0x05E5,0x0604,0x0604,0x0604,0x0603,0x0603,0x0602,0x0602,0x0601,0x0621,
0x0621,0x0620,0x0620,0x0620,0x0620,0x0E20,0x0E20,0x0E40,0x1640,0x1640,
0x1E40,0x1E40,0x2640,0x2640,0x2E40,0x2E60,0x3660,0x3660,0x3E60,0x3E60,
0x3E60,0x4660,0x4660,0x4E60,0x4E80,0x5680,0x5680,0x5E80,0x5E80,0x6680,
0x6680,0x6E80,0x6EA0,0x76A0,0x76A0,0x7EA0,0x7EA0,0x86A0,0x86A0,0x8EA0,
0x8EC0,0x96C0,0x96C0,0x9EC0,0x9EC0,0xA6C0,0xAEC0,0xAEC0,0xB6E0,0xB6E0,
0xBEE0,0xBEE0,0xC6E0,0xC6E0,0xCEE0,0xCEE0,0xD6E0,0xD700,0xDF00,0xDEE0,
0xDEC0,0xDEA0,0xDE80,0xDE80,0xE660,0xE640,0xE620,0xE600,0xE5E0,0xE5C0,
0xE5A0,0xE580,0xE560,0xE540,0xE520,0xE500,0xE4E0,0xE4C0,0xE4A0,0xE480,
0xE460,0xEC40,0xEC20,0xEC00,0xEBE0,0xEBC0,0xEBA0,0xEB80,0xEB60,0xEB40,
0xEB20,0xEB00,0xEAE0,0xEAC0,0xEAA0,0xEA80,0xEA60,0xEA40,0xF220,0xF200,
0xF1E0,0xF1C0,0xF1A0,0xF180,0xF160,0xF140,0xF100,0xF0E0,0xF0C0,0xF0A0,
0xF080,0xF060,0xF040,0xF020,0xF800];

pixels = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

function readDir() {
      var xhttp = new XMLHttpRequest();
      var dadesAJAX = "buit";
      xhttp.open('GET', 'ajaxdir', false);
```

```
      xhttp.send();
      dadesAJAX = xhttp.responseText;
      return (dadesAJAX);
}
function readArray(fname) {
      var xhttp = new XMLHttpRequest();
      var dadesAJAX = "buit";
      xhttp.open('GET', fname, false);
      xhttp.send();
      dadesAJAX = xhttp.responseText;
      return (dadesAJAX);
}
function filterFile(value, index, array) {
  return value.endsWith(".dat");
}
function getDir() {
   var dirstr = readDir();
   var dirarr = dirstr.split(",");
   var dirdat = dirarr.filter(filterFile);
   nfotos = dirdat.length;
   return dirdat;
}
function seguent() {
   if (fotonum < nfotos-1) fotonum++;
   var myVar = loadArray(arxius[fotonum]);
}
function anterior() {
   if (fotonum > 0) fotonum--;
   var myVar = loadArray(arxius[fotonum]);
}
function pescala(ctx){
  for (i=0; i<12;i++) {
    x=10*i;
    y=0;
    var tempi=(maxtemp - mintemp)/12*(i+0.5)+mintemp;
    var colorindexi= arduino_map(tempi, mintemp, maxtemp, 0, 255);
    var colori=camColors[colorindexi];    ctx.fillStyle = "#0000FF";
    ctx.fillStyle = colorRGBhex(colori);
    ctx.fillRect(x, y, 15, 20)
    }
}
function pfoto(ctx, boxSize) {
  var colorTemp;
  for (y=0; y<8; y++) {
    for (x=0; x<8; x++) {
      var val = pixels[(7-y)*8+x];
      colorTemp = val;
      if(val >= maxtemp) colorTemp = maxtemp;
      if(val <= mintemp) colorTemp = mintemp;
      var colorIndex = arduino_map(colorTemp, mintemp, maxtemp, 0, 255);
      colorIndex = arduino_constrain(colorIndex, 0, 255);
      //draw the pixels!
      var color = camColors[colorIndex];
      ctx.fillStyle = colorRGBhex(color);
      ctx.fillRect(boxSize * x, boxSize * y, boxSize, boxSize);
    }
  }
}
function loadArray(fotarx) {
  document.getElementById('nomf').innerHTML = fotarx;
  var txt= readArray(fotarx);
//  document.getElementById('crues').innerHTML = txt;
  var r = 0; var s = 0;
  r = txt.indexOf("\n");
```

```
  var fecha = txt.substring(0,r);
  document.getElementById('fecha').innerHTML = "Data: " + fecha;
  s = r+1;
  r = txt.indexOf("\n",s+1);
  var hora = txt.substring(s,r);
  document.getElementById('hora').innerHTML = "Hora: " + hora;
  s = r+1;
  r = txt.indexOf("\n",s+1);
  var tth = txt.substring(s,r);
  document.getElementById('tth').innerHTML = tth;
  s = r+1;
  r = txt.indexOf("\n",s+1);
  var rangt = txt.substring(s,r);
  document.getElementById('rangt').innerHTML = "Rang temperatures: " + rangt;
  s = r+1;
  r = txt.indexOf("\n",s+1);
  var tmin = txt.substring(s,r);
  document.getElementById('tmin').innerHTML = tmin;
  s = r+1;
  r = txt.indexOf("\n",s+1);
  var tmax = txt.substring(s,r);
  document.getElementById('tmax').innerHTML = tmax;
  for (i=0;i<64;i++) {
    s = r+1;
    r = txt.indexOf(",",s+1);
    pixels[i]= parseFloat(txt.substring(s,r));
    }
  r = rangt.indexOf("-");
  mintemp = parseFloat(rangt.substring(0,r));
  maxtemp = parseFloat(rangt.substring(r+1));
//  document.getElementById('mintemp').innerHTML = mintemp;
//  document.getElementById('maxtemp').innerHTML = maxtemp;
  var valors = "";
  for (i=0;i<8;i++) {
    for (j=0;j<8;j++) {
      valors = " " + valors + pixels[i*8+j] + ", ";
      }
    valors = valors + "<br/>";
    }
  document.getElementById('array').innerHTML = valors;

  var c = document.getElementById("escala");
  var ctx1 = c.getContext("2d");
  var esc1 = pescala(ctx1);

  var c = document.getElementById("foto");
  var ctx2 = c.getContext("2d");
  var pf1 = pfoto(ctx2, 15);
}
function arduino_map(valinp, mininp, maxinp, minout, maxout) {
   var valout = (valinp - mininp) * (maxout - minout) / (maxinp - mininp) + minout;
    valout = Math.round(valout);
    return valout;
}
function arduino_constrain(valinp, minout, maxout) {
    var valout = valinp;
    if (valinp < minout) valout = minout;
    if (valinp > maxout) valout = maxout;
    return valout;
}
function colorRGB (col565) {
   var colout = (col565 & 0xF800) * 256;
   colout = colout + (col565 & 0x07E0) * 32;
   colout = colout + (col565 & 0x001F) * 8;
```

```
   return colout;
}
function componentToHex(c) {
    var hex = c.toString(16);
    return hex.length == 1 ? "0" + hex : hex;
}
function rgbToHex(r, g, b) {
    return "#" + componentToHex(r) + componentToHex(g) +
componentToHex(b);
}
function colorR (col565) {
   var colout = (col565 & 0xF800) / 256;
   return colout;
}
function colorG (col565) {
   var colout = (col565 & 0x07E0) / 8;
   return colout;
}
function colorB (col565) {
   var colout = (col565 & 0x001F)*8;
   return colout;
}
function colorRGBhex (col565) {
   return rgbToHex(colorR(col565), colorG(col565), colorB(col565));
   return colout;
}
```

Cal dir que el programa podria haver estat molt més senzill, però he partit del mapejat de colors que faig servir al TFT 1.4 Shield de la càmera que utilitza coordenades de color especials[29] [30] (5 bits pel vermell, 6 pel verd i 5 pel blau, en total 16 bits) mentre que <canvas> fa servir 8 bits per cada component RGB.

Comentem alguns punts interessants del codi javascript:

- Les variables locals porten el prefix *var* quan es declaren:

```
  var r = 0; var s = 0;
```

- Si inicialitzem una variable sense el prefix var l'estem declarant com a variable global, accessible des de qualsevol funció:

29 Podeu obtenir més informació sobre aquestes coordenades de color i la llibreria gràfica GFX d'Adafruit a https://learn.adafruit.com/adafruit-gfx-graphics-library/coordinate-system-and-units

30 El sistema de mapejat de temperatura a colors és part dels exemples de la llibreria Adafruit pel sensor de càmera tèrmica AMG8833, ho podeu veure a https://learn.adafruit.com/adafruit-amg8833-8x8-thermal-camera-sensor/arduino-thermal-camera

```
pixels = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
```

- Com veieu, no és fàcil declarar arrays amb una mida fixa. Per això és tan llarga la declaració anterior. En canvi és fàcil afegir nous elements a un array amb el mètode *push*.

- És molt senzill generar un array a partir d’un text on els valors estan separats per un delimitador (el directori AJAX dona els noms d’arxius separats per comes:

```
    var dirarr = dirstr.split(",");
```

- I podem eliminar els valors que no compleixen certa condició (jo només vull noms d’arxius de fotografies) definida en una funció pròpia:

```
    var dirdat = dirarr.filter(filterFile);
```

- Podem modificar el *canvas*[31] des de qualsevol funció amb el seu *id*:

```
  var c = document.getElementById("foto");
  var ctx2 = c.getContext("2d");
```

- O amb el descriptor obtingut amb *getContext()*:

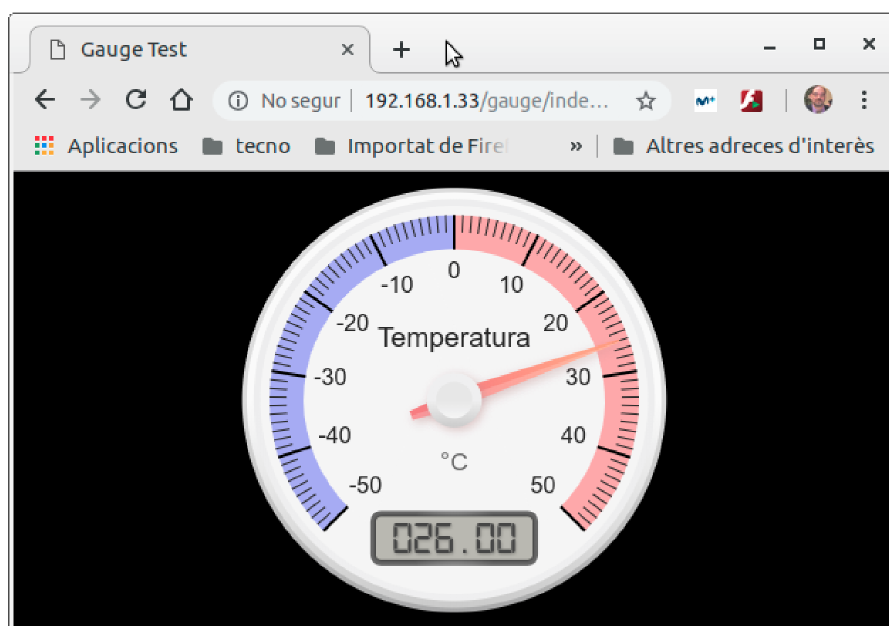
```
      ctx.fillStyle = colorRGBhex(color);
      ctx.fillRect(boxSize * x, boxSize * y, boxSize, boxSize);
```

31 Trobareu més informació sobre *canvas* i les seves funcions a https://www.w3schools.com/graphics/canvas_intro.asp

La llibreria gràfica gauge.js

Es tracta d'una llibreria gràfica per mostrar valors d'un sensor- Té una llicencia MIT[32] que en permet desar-la a la nostra SD virtual i adaptar-la a les nostres necessitats.

La darrera versió de la llibreria sembla més fàcil de configurar i utilitzar, però a mi m'agrada molt una versió més antiga pel seu aspecte més formal:



A l'exemple *gauge* la faig servir amb el DHT shield:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>
#include <WEMOS_DHT12.h>
DHT12 dht12;
```

32 Més informació sobre la llibreria gauge.js a http://bernii.github.io/gauge.js/ on fins-i-tot podeu provar en línia variacions de paràmetres a la darrera versió de la llibreria

```
const char *ssid = "nom de la xarxa";
const char *password = "contrasenya";

ESP8266WebServer server ( 80 );
const int led = BUILTIN_LED;
int t;

void handleRoot() {
  digitalWrite ( led, LOW );
  char temp[400];
  int sec = millis() / 1000;
  int min = sec / 60;
  int hr = min / 60;
  snprintf ( temp, 400,
"<html>\
  <head>\
    <meta http-equiv='refresh' content='5'/>\
    <title>ESP8266 Demo</title>\
    <style>\
      body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-
Serif; Color: #000088; }\
    </style>\
  </head>\
  <body>\
    <h1>Sonda de temperatura amb ESP8266</h1>\
    <p><a href='gauge/index.html'>gauge</a><p/>\
    <p>Uptime: %02d:%02d:%02d</p>\
  </body>\
</html>",
    hr, min % 60, sec % 60
  );
  server.send ( 200, "text/html", temp );
  digitalWrite ( led, HIGH );
}

void handleNotFound() {
  digitalWrite ( led, LOW );
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += ( server.method() == HTTP_GET ) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for ( uint8_t i = 0; i < server.args(); i++ ) {
    message += " " + server.argName ( i ) + ": " + server.arg ( i ) + "\n";
  }
  server.send ( 404, "text/plain", message );
  digitalWrite ( led, HIGH );
}

void handleAjax(){
  digitalWrite ( led, LOW );
  dht12.get();
  t=dht12.cTemp;
  Serial.print("Temperature: ");
  Serial.println(t);
  String out = "";
  char temp[100];
  snprintf(temp, 100, "%d", t);
  out += temp;
  server.send ( 200, "text/txt", out);
  digitalWrite ( led, HIGH );
```

```
}

void setup ( void ) {
  pinMode ( led, OUTPUT );
  digitalWrite ( led, HIGH );
  Serial.begin ( 115200 );
  WiFi.begin ( ssid, password );
  SPIFFS.begin();
  Serial.println ( "" );
  // Wait for connection
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
    Serial.print ( "." );
  }
  Serial.println ( "" );
  Serial.print ( "Connected to " );
  Serial.println ( ssid );
  Serial.print ( "IP address: " );
  Serial.println ( WiFi.localIP() );
  server.on ( "/", handleRoot );
  server.serveStatic("/gauge", SPIFFS, "/gauge"); //  gauge files dir
  server.on ("/ajax_info.txt", handleAjax );
  server.onNotFound ( handleNotFound );
  server.begin();
  Serial.println ( "HTTP server started" );
}
void loop ( void ) {
  server.handleClient();
}
```

Podem configurar el control al fitxer */gauge/index.html*:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <meta name="viewport" content="width=device-width">
  <title>Gauge Test</title>
  <script src="gauge.js"></script>
  <style>body{padding:0;margin:0;background:#222}</style>
  <script>
function loadTemp() {
  var xhttp = new XMLHttpRequest();
  var temp = 0;
  xhttp.open("GET", "../ajax_info.txt", false);
  xhttp.send();
  temp = parseFloat(xhttp.responseText);
  return (temp);
}
function DrawGauge()
{
  gauge = new Gauge({
  renderTo    : 'gauge',
  width       : document.body.clientWidth,
  height      : document.body.clientHeight,
  glow        : true,
  units       : '°C',
  title       : 'Temperatura',
  minValue    : -50,
  maxValue    : 50,
  majorTicks  : ['-50','-40','-30','-20','-10','0','10','20','30','40','50'],
  minorTicks  : 10,
  strokeTicks : false,
  highlights  : [
```

```
           { from : -50, to : 0, color : 'rgba(0,   0, 255, .3)' },
           { from : 0, to : 50, color : 'rgba(255, 0, 0, .3)' }
           ],
  colors : {
    plate      : '#f5f5f5',
    majorTicks : '#000',
    minorTicks : '#222',
    title      : '#222',
    units      : '#666',
    numbers    : '#222',
    needle     : { start : 'rgba(240, 128, 128, 1)', end : 'rgba(255, 160, 122, .9)' }
            },
  animation : {
                        delay : 25,
                        duration: 500,
                        fn : 'bounce'
                        }
  });
gauge.onready = function() {
            setInterval( function() {
            gauge.setValue( loadTemp());
            }, 1000);
  };
            gauge.draw();
};
  </script>
  </head>
  <body style="background-color:black;">
            <canvas id="gauge"></canvas>
            <div id="console"></div>
            <script>
                        DrawGauge();
            </script>
</html>
```

Com veieu és molt fàcil canviar

```
  units        : '°C',
  title        : 'Temperatura',
  minValue     : -50,
  maxValue     : 50,
```

o d'altres paràmetres.

La funció que dona les dades AJAX es podria reduir a la seva expressió mínima:

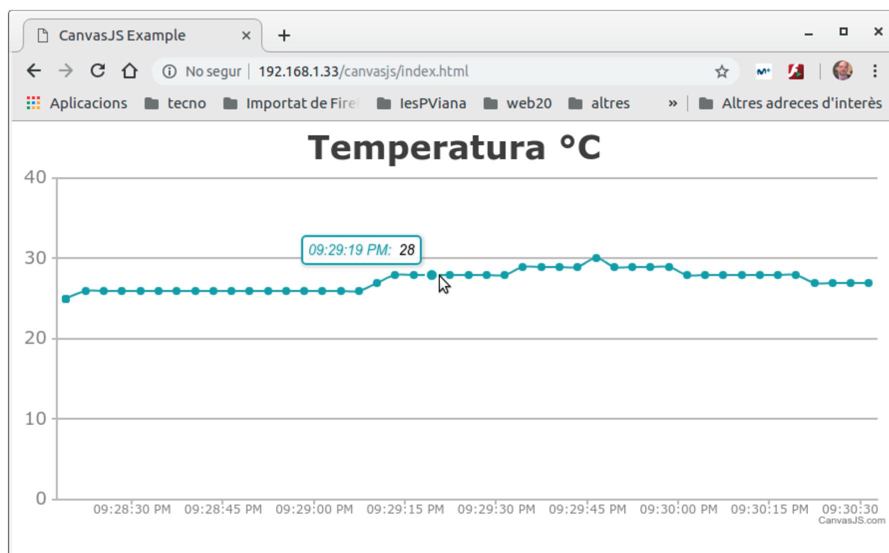
```
void handleAjax(){
  dht12.get();
  t=dht12.cTemp;
  String out = String(t);
  server.send ( 200, "text/txt", out);
}
```

La llibreria gràfica *canvasjs*. Gràfiques de dades.

Aquesta llibreria[33] la podeu fer servir amb una llicència CC BY-NC.

A la carpeta *canvasjs/examples* trobareu diferents exemples de gràfiques que es poden fer amb aquesta llibreria.

El fitxer *canvas/index.html* és una versio modificada de l'exemple *spline*, que actualitza les dades del nostre DHT shield amb tecnologia AJAX:



El codi arduino és el mateix que el del capítol La llibreria gràfica gauge.js, ja que el vaig fer compatible amb els dos exemples de les llibreries. El que canvia en aquest exemple *canvasjs* és el contingut de la SD virtual. No només hi ha aquest tipus de gràfica. A la seva web podem trobar més de 30 exemples: àrees, rangs, acumulats, columnes, barres, sectors, bombolles, capses ... El problema és que per fer servir qualsevol gràfic ha d'utilitzar codi extern al núvol i, per tant, tenir connexió a internet. No funciona em mode AP!

33 Disponible a http://canvasjs.com

Veiem el contingut de *canvas/index.html*:

```
<!DOCTYPE HTML>
<html>
<head>
  <script type="text/javascript">
  window.onload = function () {
  var chart = new CanvasJS.Chart("chartContainer", {
    title: {
      text: "Temperatura °C"
             },
    data: [{
      type: "spline",
      dataPoints: [
         { x: new Date() , y: 25 },
         { x: new Date() , y: 25 },
         { x: new Date() , y: 25 },
         { x: new Date() , y: 25 },
         { x: new Date() , y: 25 }
              ]
     }]
  });
  chart.render();

  function loadTemp() {
    var xhttp = new XMLHttpRequest();
    var temp = 0;
    xhttp.open("GET", "../ajax_info.txt", false);
    xhttp.send();
    temp = parseFloat(xhttp.responseText);
    chart.options.data[0].dataPoints.push({ x: new Date() , y:   temp});
    chart.render();
  }
  setInterval( function() {
            loadTemp();
            }, 3000);
            }
  </script>
  <script src="canvasjs.min.js"></script>
  <title>CanvasJS Example</title>
</head>
<body>
  <div id="chartContainer" style="height: 400px; width: 100%;"></div>
</body>
</html>
```

Part VII. Projectes

La veritat és que els meus alumnes i jo gaudim fent projectes. Molt!

La primera fase en un projecte es demostrar la seva viabilitat: tenir un prototip operatiu. El nostre kit és perfecte per a aquesta missió, amb la seva diversitat de shields i la possibilitat de connectar components externs.

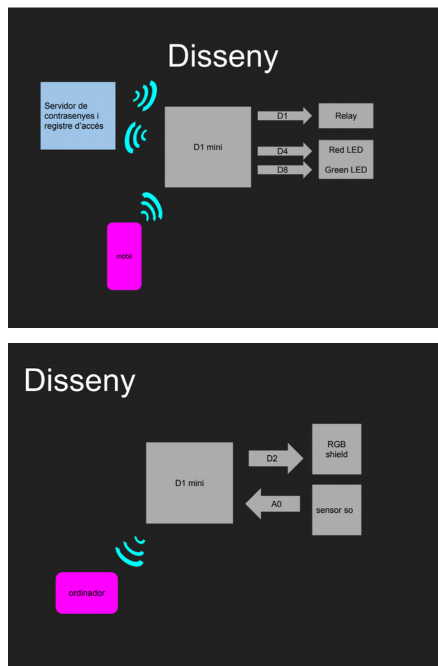
Depenent del temps disponible ens podem permetre el luxe de fer una segona fase: el disseny d'un prototip comercial. Optimitzarem el disseny, normalment prescindint de la nostra base triple, posarem shields de connectors (podem utilitzar el nostre Shield de connexions (JORTS) personalitzat o un dels seus derivats estàndard Shield CON1 o Shield CON2) pels components externs, afegirem una font d'alimentació adient, li farem una capsa que provarem amb la impressora 3D... Al capítol Part VII. Projectes ja en parlarem.

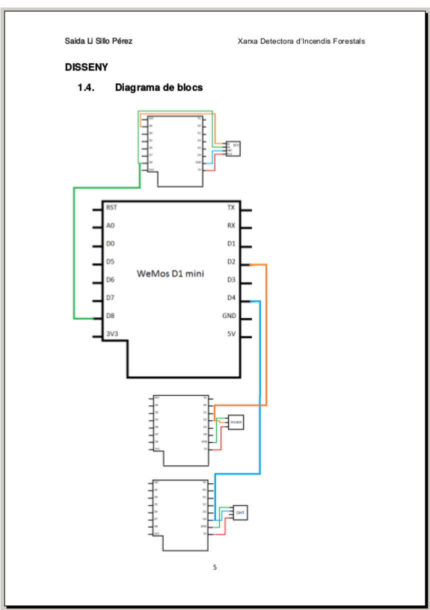
Quan fem un projecte el primer que hem de fer és pensar quins sensors i actuadors necessitarem, i com serà la interfície amb l'usuari. Aquesta última pot incloure tant maquinari com control remot. Concretarem aquests elements en un diagrama de blocs, on detallarem els pins que utilitza cada component. També pot ser interessant fer una graella[34] amb els pins del D1 mini i quins shields o components externs van connectats. Sigui com sigui, la idea es tenir centralitzada la informació de qu**è** tenim connectat a on.

Veiem alguns exemples dels meus alumnes[35]:

34 Al Annex 10: Plantilla connexions projecte D1 teniu una plantilla de graella que podeu utilitzar als vostres projectes

35 Podeu accedir a informació més detallada d'aquests projectes a https://github.com/jorts64/kit-D1-mini/tree/master/projectes





Taula 4

Àmbit Lleure
Escultura música i color

Connexions D1 mini

D1 mini	Shield / mòdul	Notes
D0		
D1 / SCL		
D2 / SDA	RGB	Bloqueja ús I2C
D3		
D4		
D5	PB1 MP3 TX	
D6	PB1 MP3 RX	
D7		
D8		
A0		

Documentació

https://wiki.wemos.cc/products:d1_mini_shields:ws2812b_rgb_shield

https://sites.google.com/a/iepegasoviana.cat/sensors-arduino/actuadors/serial-mp3-player

Taula 2

Àmbit mobilitat
Control aforament pàrquing

Connexions D1 mini

D1 mini	Shield / mòdul	Notes
D0		
D1 / SCL	OLED	I2C
D2 / SDA	OLED	I2C
D3	PB1 SR1	Inc
D4	PB2 SR2	Dec
D7	PB3 4 dígits CLK	
D8	PB3 4 dígits DIO	
D5	PB4 Led R	
D6	PB4 Led G	
A0		

Documentació

https://wiki.wemos.cc/products:d1_mini_shields:oled_shield

https://sites.google.com/a/iepegasoviana.cat/sensors-arduino/actuadors/display-tm1637-4-digits

Aneu amb compte amb el darrer exemple: van tenir molts problemes amb aquestes assignacions. De fet aquest grup d'alumnes van fer genuïna recerca, trobant les limitacions d'alguns pins amb el tipus d'entrada.

Una vegada definit el maquinari primer provarem cada shield o component extern per separat amb un programa de test. Podeu utilitzar els que us he mostrat per cada shield, o tal vegada alguna cosa més senzilla. És important desar aquests programes, que recomano inserir a la memòria com a annexos, per poder provar en qualsevol moment que un component funciona correctament.

Després connectarem tot el maquinari i tornarem a passar cada programa de test per comprovar que en connectar tot el maquinari no hi ha cap tipus de problema, com ara col·lisions entre els pins. Per exemple, a la documentació oficial del TFT 1.4 Shield en cap moment s’esmena que el pin D6 queda afectat, però així és.

Amb tot connectat i testejat farem un darrer programa de test que comprovi tot el maquinari. Això ens servira per veure com fusionem tots els codis d’exemple en un de sol: llibreries i variables al principi, inicialitzacions al *setup()* i comandes al *loop()*.

Finalment ja podrem treballar en el codi propi del nostre projecte, donant-li la funcionalitat desitjada.

Bona sort!

Part VIII. Prototips comercials

El nostre prototip de viabilitat que utilitza el nostre kit funciona. Ara el volem optimitzar, pensant en la seva comercialització.

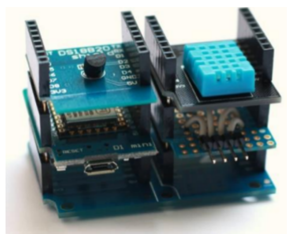
Tria de la base. Connexions mòduls externs

El primer que hem de fer es decidir quina base fem servir, tenint en compte si hem de connectar mòduls externs. Tenim diferents opcions:

Cap base.	Interessant quan tenim pocs shields i connexions externes.	Fem servir el propi D1 mini com a base soldant terminals femella. Caldrà un shield de connectors per a les connexions a mòduls externs.
Base doble	Interessant quan tenim un nombre modest de shields i connexions externes.	Podem fer servir una de les columnes per connectar terminals mascle[36], reproduint la tercera columna de la nostra base triple, per les connexions a mòduls externs, o fer servir shields de connectors.
Base triple	Interessant quan tenim un nombre alt de shields i connexions externes.	Mateixes opcions que en el case de la base doble.

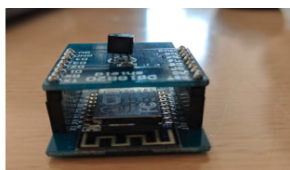
Veiem alguns exemples de prototips dissenyats pels meus alumnes. Penseu que alguns es podrien avui fer més senzills amb el nou shield de connectors:

36 Aneu amb compte amb la versió de base doble. La versió antiga només només portava els connectors per a cada columna. L'actual (V2.0.0) és semblant a la triple base, amb línies de potència i zona de prototipatge.

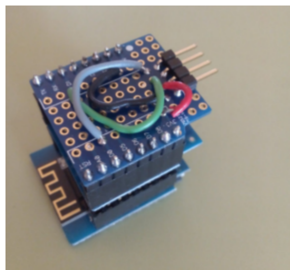


Node sensor d'incendis
2017-18 TR Xarxa detectora d'incendis[37]
D1 mini / protoboard connector sensor flama / DHT / DS18B20

Avui podríem reemplaçar la protoboard per un Shield de connexions (JORTS).

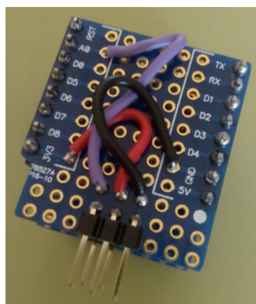


Sensor nevera
PRE 2017-18 IoT[38]



Conjunt per a taula
PRE 2016-17 IoT[39]
D1 mini / SHT30 / connector I2C sensor llum.

Avui podriem reemplaçar la protoboard del connector per una Shield CON2



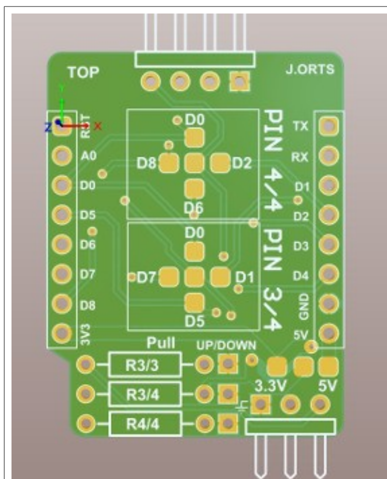
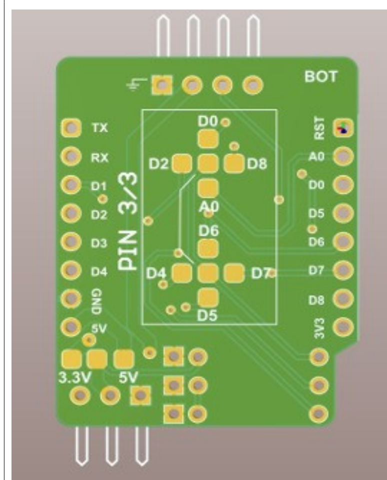
Conjunt per a finestra
PRE 2016-17 IoT_hospital
D1 mini / 2 protoboards amb connectors DS18B20 , sensor magnètic finestra, LDR.

Avui podríem reemplaçar les protoboards amb per una única Shield de connexions (JORTS), amb una configuració especial.

37 Podeu accedir a informació més detallada d'aquest projecte a https://github.com/jorts64/kit-D1-mini/tree/master/projectes/2017-18%20TR
38 Podeu accedir a informació més detallada d'aquest projecte a https://github.com/jorts64/kit-D1-mini/blob/master/projectes/2017-18%20PRE4t/nevera.pdf
39 Podeu accedir a informació més detallada d'aquest projecte a https://github.com/jorts64/kit-D1-mini/blob/master/projectes/2016-17%20PRE4t/IoT_hospital.pdf

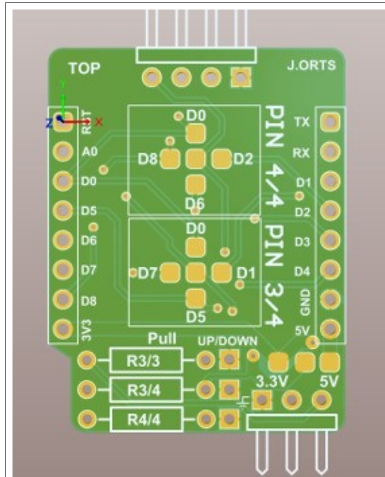
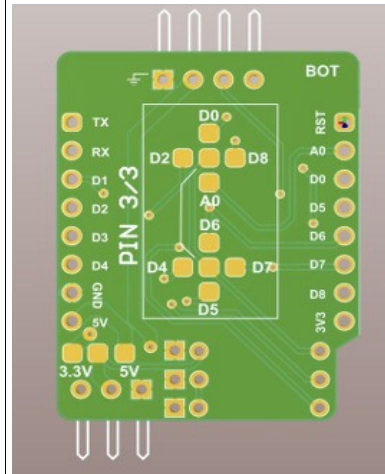
Recordeu que per les connexions a mòduls externs tenim disponible el Shield de connexions (JORTS), molt flexible. El meu nebot Javi i jo n’estem molt orgullosos d’aquest disseny. Fixeu-vos quants d’aquests dissenys es podrien haver fet amb aquesta shield.

Veiem la seva aplicació al primer projecte. Volem connectar un sensor de flama, amb sortides digital i analògica:

<table>
<tr>
<td>
</td>
<td>Fem un pont al selector 4/4 entre D8 i el mig</td>
</tr>
<tr>
<td>
</td>
<td>Fem un pont al selector 3/3 entre A0 i el mig
Fem un pont entre 3,3V i el mig

Ara ja podem connectar el senyal digital a D8 de J4 i la resta de connexions a J3</td>
</tr>
</table>

Veiem la seva aplicació al darrer projecte. Volem connectar un DS18B20 que no porta pull-up , un sensor digital i una LDR a un únic shield:

	Fem un pont al selector 4/4 entre D6 i el mig Fem un pont al selector 3/4 entre D7 i el mig Posem una resistència de 4k7 a R4/4 connectada a UP Posem una resistència de 10k a R3/4 connectada a UP Posem una resistència de 100k a R3/3 connectada a DOWN Fem un pont entre 3,3V i el mig
	Fem un pont al selector 3/3 entre A0 i el mig Fem un pont entre 3,3V i el mig Ara ja podem connectar: - DS18B20 al connector J3 - Sensor digital entre pins 3 i 1 de J4 - LDR entre pins 4 i 2 de J4

Realment ens va quedar un disseny molt eficient i flexible del shield de connexions!

Disseny de la capsa

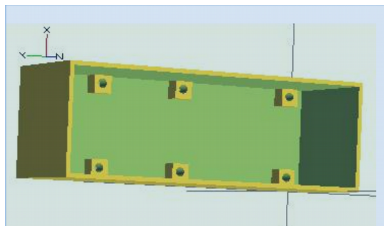
Naturalment voldrem una capsa pel nostre prototip. Res més senzill amb les impressores 3D avui en dia al nostre abast.

Tenim accés a molt programari per dissenyar en 3D. Jo personalment recomano OpenSCAD o qualsevol de les seves variants: ScorchCAD (Android), OpenJSCAD (en línia), BlocksCAD (blocs en línia) ...

Aquí tenim com a exemple part del disseny en OpenSCAD que les meves alumnes Núria Padilla i Judit Casas van fer pel seu TR:

```
include<M3.scad>
module capsa_1_v3(){
difference() {
translate([0,-5,0])
cube([39.4,114.3,43.5]);
translate([1.5,-3.5,1.5])
cube([36.4,111.3,43.5]);
}
difference(){
translate ([1.5,1.5,1.5])
cube([6.5,6.5,6]);
union () {
translate ([5,5,1.5]) forat_M3(10);
}
}
difference(){
translate ([31.5,1.5,1.5])
cube([6.5,6.5,6]);
translate ([35,5,1.5]) forat_M3(10);
}
difference(){
translate ([31.5,45.5,1.5])
cube([6.5,6.5,6]);
translate ([35,49,1.5])forat_M3(10);
}
difference(){
translate ([1.5,45.5,1.5])
cube([6.5,6.5,6]);
translate ([5,49,1.5]) forat_M3(10);
}
difference(){
translate ([1.5,77.5,1.5])
cube([6.5,6.5,6]);
translate ([5,81,1.5]) forat_M3(10);
}
difference(){
translate ([1.5,100.5,1.5])
cube([6.5,6.5,6]);
translate ([5,104,1.5])forat_M3(10);
}
difference(){
translate ([31.5,77.5,1.5])
cube([6.5,6.5,6]);
translate ([35,81,1.5]) forat_M3(10);
}
difference(){
translate ([31.5,100.5,1.5])
cube([6.5,6.5,6]);
translate ([35,104,1.5]) forat_M3(10);
}
}
```

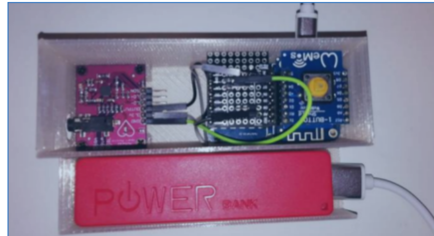
Podeu veure una imatge d’aquest part del disseny:



Com veieu elles van optar per fer un disseny modular. En el seu cas era molt intel·ligent: havien de fer dos capses molt semblants amb petites diferències.

El disseny total (el seu TR és el disseny d'un holter econòmic que porta el D1 mini) és molt més complicat. Us estalvio els passos i us mostro el resultat final:





Càlcul del pressupost

Aquest tema sempre els hi costa als meus alumnes.

Per fer un pressupost realista caldria fer-ne tres:

- Un pressupost del prototip de viabilitat, tenint en compte la mà d'obra i el preu del material
- Un pressupost del prototip comercial, tenint en compte la mà d'obra i el preu del material (únicament dels extres originats per aquesta adaptació).
- Un pressupost d'una sèrie (posem 1000 unitats) on es tingui en compte únicament el que costaria fer aquestes 1000 noves unitats (mà d'obra + material) i al qual hem d'afegir la part proporcional (1/1000 en aquest cas) dels dos pressupostos anteriors.

Part IX. Actualització WiFi del programa (OTA)

Si tenim un microcontrolador com l'ESP8266 amb WiFi, per què hem d'endollar un cable per actualitzar el programa?

L'ESP8266 ens permet la programació OTA (On The Air). És molt senzill. Només cal copiar el fitxer *OTA.cpp*[40] dintre de la carpeta del programa, declarar la funció *OTAtest()* i cridar-la quan ens interessi.

No oblideu canviar *OTAssid* i *OTApassword* al principi del fitxer *OTA.cpp* pels valors adients, encara que sigui una xarxa generada pel vostre mòbil :

```
const char* OTAssid = "esp8266AP";
const char* OTApassword = "robotica";
```

He volgut posar un nom de xarxa i contrasenya diferents per temes de seguretat. A un espai puc tenir el meu D1 mini connectat a la xarxa local. Però si vull actualitzar el programa portaré el meu portàtil, el meu mòbil, generaré amb el mòbil una xarxa que només serveix per a aquest menester i així evitaré que altres persones puguin hackejar el meu sistema. Naturalment a casa no em complico la vida i poso els valors de la meva xarxa local.

Veiem l'exemple *OTAtest*:, que activa OTA en prémer el botó del 1-button shield:

```
const int buttonPin = D3;
const int ledPin = BUILTIN_LED;

void OTAwait();

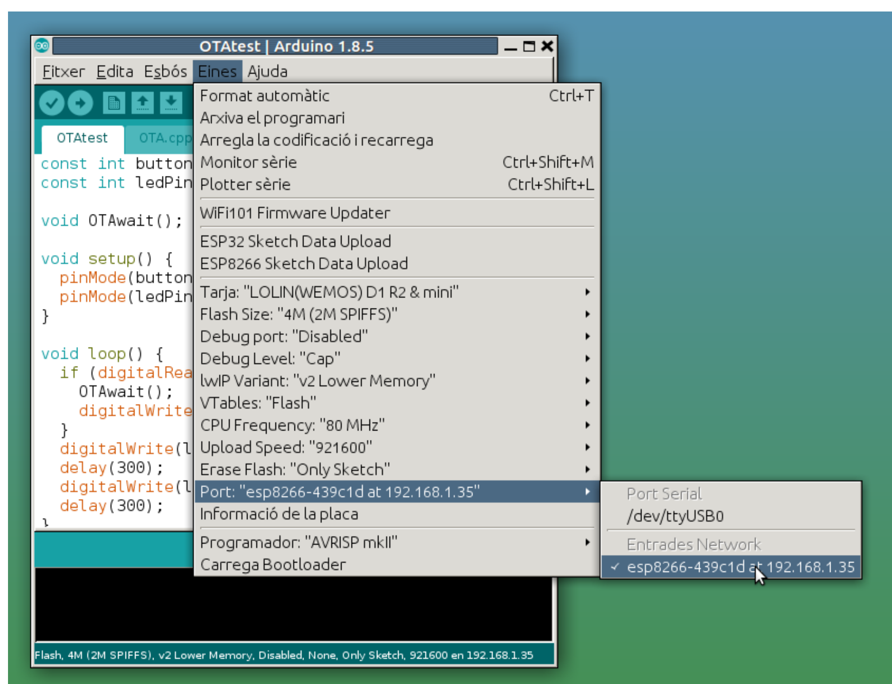
void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  if (digitalRead(buttonPin) == LOW){
```

40 El contingut del fitxer *OTA.cpp* el teniu disponible a

```
    OTAwait();
    digitalWrite(ledPin, HIGH);
  }
  digitalWrite(ledPin, HIGH);
  delay(300);
  digitalWrite(ledPin, LOW);
  delay(300);
}
```

Una vegada carregat el programa, només caldrà prémer el botó. El led deixa de fer pampallugues per quedar fix esperant l'actualització. Tanquem i obrim l'arduino IDE per que reconegui el dispositiu, que trobarem a l'opció Eines→ Port dintre del apartat Entrades Network. El seleccionem i ja podem reprogramar el nostre D1 mini via WiFi.



Part X. Altres plaques basades en l’ESP8266

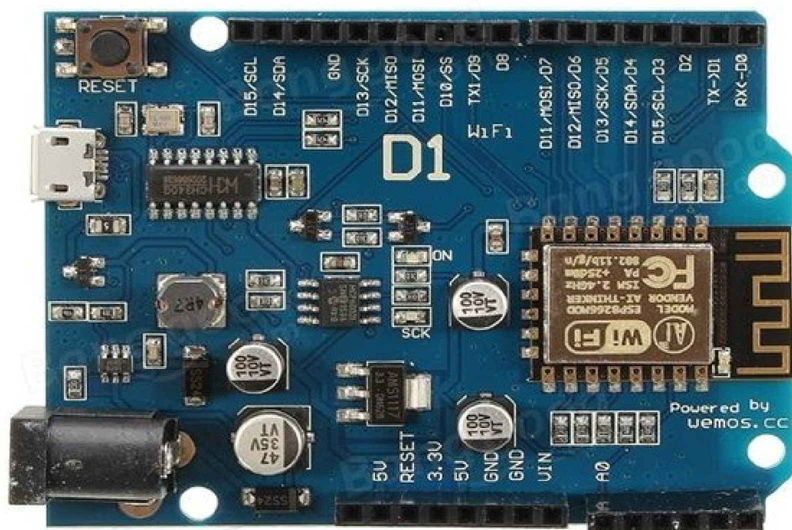
El nostre kit fa servir el D1 mini, però no hem de limitar els nostres coneixements a aquesta placa. Podem aprofitar els nostres coneixements a qualsevol placa amb l’ESP8266.

La placa D1. Un ESP8266 amb factor de forma Arduino UNO

Com el seu nom indica la nostra D1 mini va evolucionar a partir d’aquesta placa que porta un ESP8266 i reprodueix la placa Arduino UNO. Potser interessant per a algun projecte, però jo no la recomano. La trobo conflictiva: mapejat dels pins discutible, sense adaptació de nivells 3,3 V / 5 V ... Ningú ja en parla. Tothom parla de la D1 mini!

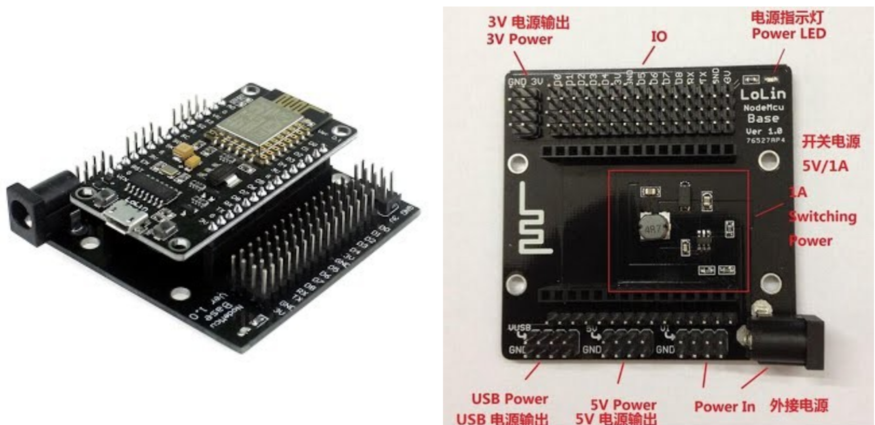






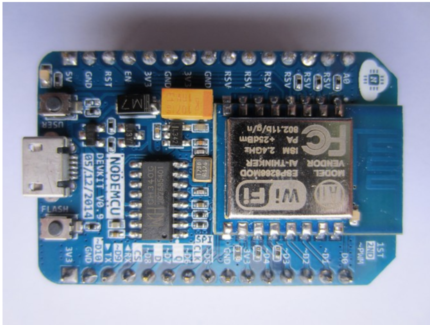
Els mòduls ESP12 V1 i V3. La placa NODEMCU Base v1.0

El mòdul és més ample que el que veurem a Els mòduls ESP12 V2. La placa NODEMCU motor shield i porta un xip USB-UART CH341 (rectangular). Podem utilitzar directament el mòdul amb una breadboard gràcies a les seves potes.



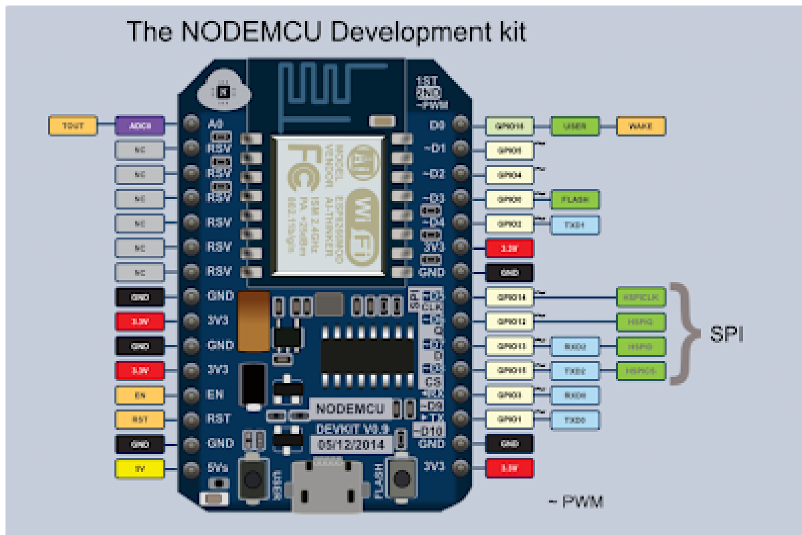
La placa porta 4 blocs d'alimentació (porta un xip regulador 5V 1A) i 4 pins per a cada entrada/sortida del costat dret (només 1 per l'esquerre, ja que a la V1 només calia el RST).

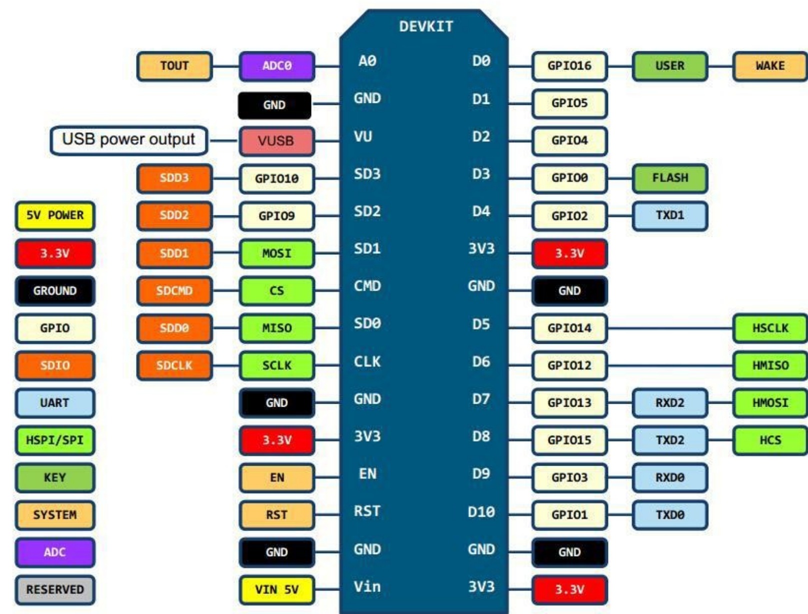
La V1 es programa a l'arduino IDE seleccionant la placa NODEMCU 0.9. En canvi, la V3, que té la mateixa mida però més pins assignats, correspon a la placa NODEMCU 1.0 de l'arduino IDE.





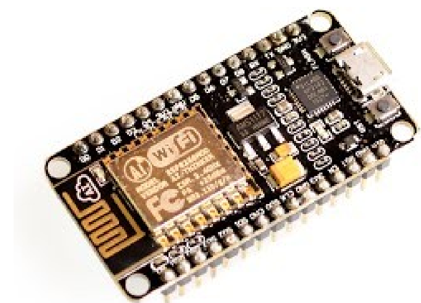
Encara que comparteixen factor de forma i tots dos mòduls són compatibles amb la mateixa placa, fixeu-vos que hi ha diferencies en el pin-out:

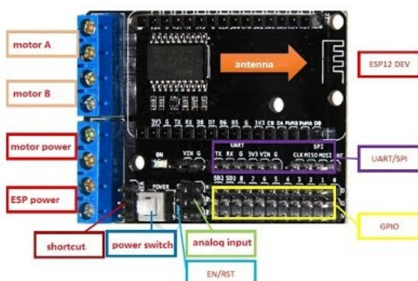



PIN DEFINITION
DEVKIT
A0
GND
VU
SD3
SD2
SD1
CMD
SD0
CLK
GND
3V3
EN
RST
GND
Vin
D0
D1
D2
D3
D4
3V3
GND
D5
D6
D7
D8
D9
D10
GND
3V3
TOUT
ADC0
GND
USB power output
VUSB
SDD3
GPIO10
SDD2
GPIO9
SDD1
MOSI
SDCMD
CS
SDD0
MISO
SDCLK
SCLK
GND
3.3V
EN
RST
GND
VIN 5V
GPIO16
USER
WAKE
GPIO5
GPIO4
GPIO0
FLASH
GPIO2
TXD1
3.3V
GND
GPIO14
HSCLK
GPIO12
HMISO
GPIO13
RXD2
HMOSI
GPIO15
TXD2
HCS
GPIO3
RXD0
GPIO1
TXD0
GND
3.3V
5V POWER
3.3V
GROUND
GPIO
SDIO
UART
HSPI/SPI
KEY
SYSTEM
ADC
RESERVED
D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

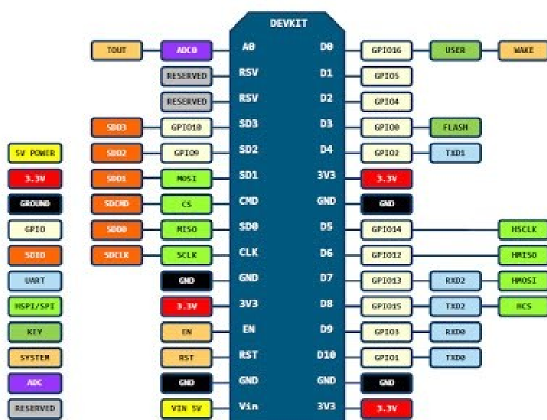
Els mòduls ESP12 V2. La placa NODEMCU motor shield

El mòdul és més estret que el que hem vist a Els mòduls ESP12 V1 i V3. La placa NODEMCU Base v1.0 , i porta un xip USB-UART CP210x (quadrat).





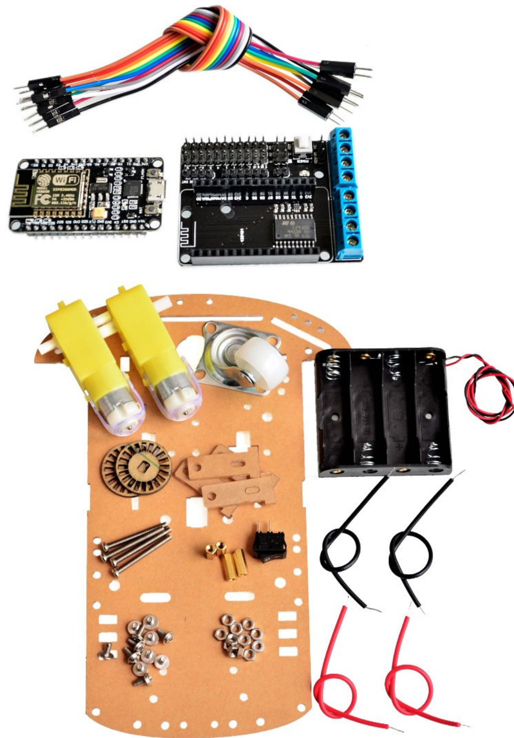
Ajusta amb una placa de control de motors amb xip L293D perfecta per controlar robots, amb els pins 5,4,0 i 2 (D1, D2, D3, D4). Els pins de sortida porten la típica organització compatible amb servo amb l'alimentació al mig, en aquest cas de 3,3V. A més a més porta un interruptor per apagar el conjunt.



Realment és un conjunt molt interessant ja que amb un preu molt econòmic tenim tot el necessari per implementar un robot.

Un exemple el trobareu a https://github.com/squix78/esp8266-projects/tree/master/arduino-ide/wifi-car. El fet d'ocupar D1 i D2 ens obligarà a desplaçar SDA i SCL si volem utilitzar el protocol I2C.

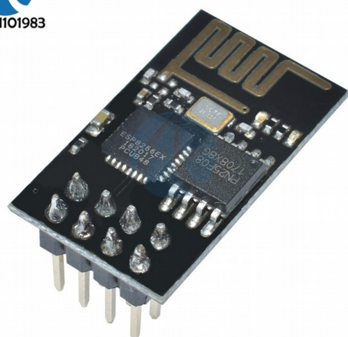
De fet, trobaren packs amb aquesta placa i l’estructura bàsica del robot a un preu molt raonable.

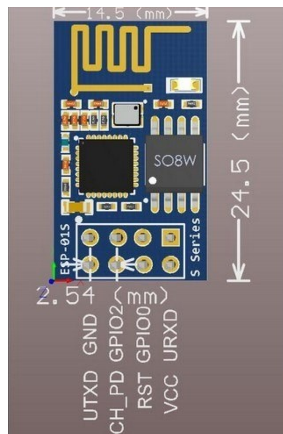


Els mòduls ESP-01. Plaques compatibles

ESP-01 és una placa amb un ESP8266 amb 512kB o 1MB de memòria[41] i un nombre molt reduït de connexions.







Per programar aquest mòduls recomano aquesta placa, que s'endolla directament a l'ordinador i porta un commutador per canviar a mode programació:





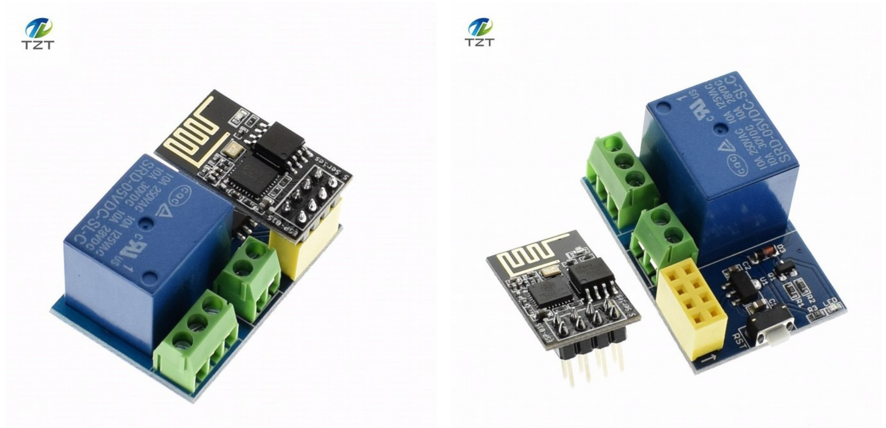
Trobarem un bon grapat de plaques que utilitzen aquests mòduls:

41 Podeu fer servir a l'IDE Arduino *Exemples → ESP8266 → CheckFlashConfig* per determinar la memòria real del vostre ESP-01

ESP-01 relay module

Aquesta placa permet controlar un relé amb el pin GPIO0.

S’ha d’alimentar amb 5V.



Veiem un exemple d’implementació AJAX (com a la resta d’exemples d’aquest annex, cal copiar els fitxers *FSManager.h* i *FSManager.cpp* a la mateixa carpeta[42]):

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;
#define PIN 0

int estat;

void handleRoot() {
  server.send(200, "text/plain", "hello from esp8266!");
}


void engegar() {
  digitalWrite(PIN, LOW);
  estat=1;
  server.send(200, "text/plain", "ON");
}

void apagar() {
  estat=0;
  server.send(200, "text/plain", "OFF");
  digitalWrite(PIN, HIGH);
}
```

42 Veure Annex 9: Afegint la funcionalitat de FSManager als nostres projectes

```
void canviar() {
  if(estat==0){
    digitalWrite(PIN, LOW);
    estat=1;
    }
   else{
    digitalWrite(PIN, HIGH);
    estat=0;
    }
  server.send(200, "text/plain", "CHANGED");
}

void report() {
  String t = String(estat);
  server.send(200, "text/plain", t);
}

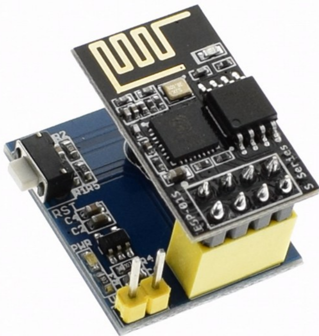
void setup(void){
  pinMode(PIN, OUTPUT);
  digitalWrite(PIN, HIGH);
  estat = 0;
  SPIFFS.begin();
  initHelper(); //inclou connexió Wifi i server
  server.on("/", handleRoot);
  server.on("/on", engegar);
  server.on("/off", apagar);
  server.on("/change", canviar);
  server.on("/estat", report);
  server.begin();
  }

void loop(void){
  espera(100);
  }
```

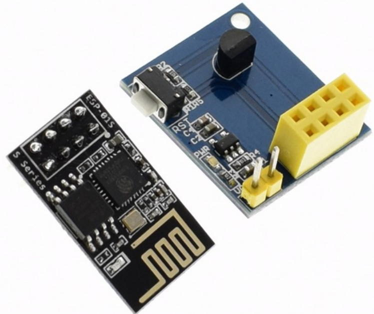
DS18B20 Temperature Sensor Module

Aquesta placa permet mesurar la tempertura amb el pin GPIO2. Es pot alimentar amb 3,7 V : 12 V, admeten bateries de Li de 3,7 V.









Veiem un exemple d’implementació AJAX:

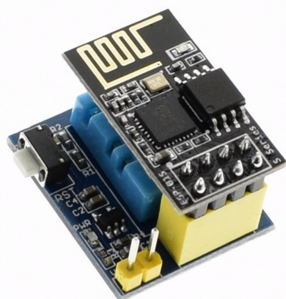
```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 2  // DS18B20 pin
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature DS18B20(&oneWire);
void handleRoot() {
  server.send(200, "text/plain", "hello from esp8266!");
}

void temperatura() {
  DS18B20.requestTemperatures();
  float t = DS18B20.getTempCByIndex(0);
  String s = String(t);
  server.send(200, "text/plain", s);
}
void setup(void){
  DS18B20.begin();
  SPIFFS.begin();
  initHelper(); //inclou connexió Wifi i server
  server.on("/", handleRoot);
  server.on("/temp", temperatura);
  server.begin();
  }
void loop(void){
  espera(100);
  }
```

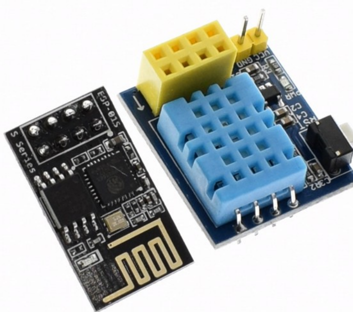
ESP-01 DHT11 Temperature Humidity Sensor Module

Aquesta placa permet mesurar la tempertura i la humitat amb el pin GPIO2. Es pot alimentar amb 3,7 V : 12 V, admeten bateries de Li de 3,7 V. Porta un DHT11 amb un protocol d’un sol fil, diferent al del nostre kit, i que utilitza la llibreria *DHT sensor*.









Veiem un exemple d’implementació AJAX:

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;
#include "DHT.h"
#define DHTTYPE DHT11   // DHT 11
#define DHTPIN 2     // what digital pin we're connected to
DHT dht(DHTPIN, DHTTYPE);

void handleRoot() {
  server.send(200, "text/plain", "hello from esp8266!");
}


void humitat() {
  float h = dht.readHumidity();
  String s = String(h);
  server.send(200, "text/plain", s);
}

void temperatura() {
  float t = dht.readTemperature();
  String s = String(t);
  server.send(200, "text/plain", s);
}
```

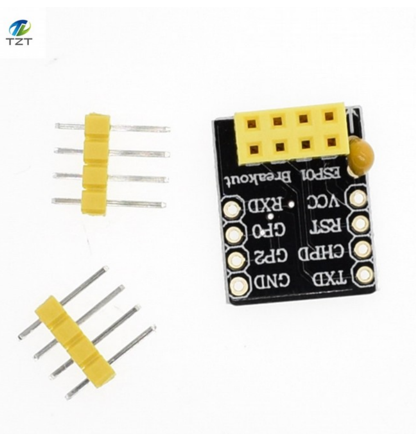
```
void setup(void){
  dht.begin();
  SPIFFS.begin();
  initHelper(); //inclou connexió Wifi i server
  server.on("/", handleRoot);
  server.on("/rh", humitat);
  server.on("/temp", temperatura);
  server.begin();
  }

void loop(void){
  espera(100);
  }
```

ESP-01 Breakout Module

Aquesta placa és interessant per fer mini-projectes amb l'ESP-01, ja que ens dona accés a totes les potes de l'ESP-01.





ESP-01 WS2812 RGB LED Controller module

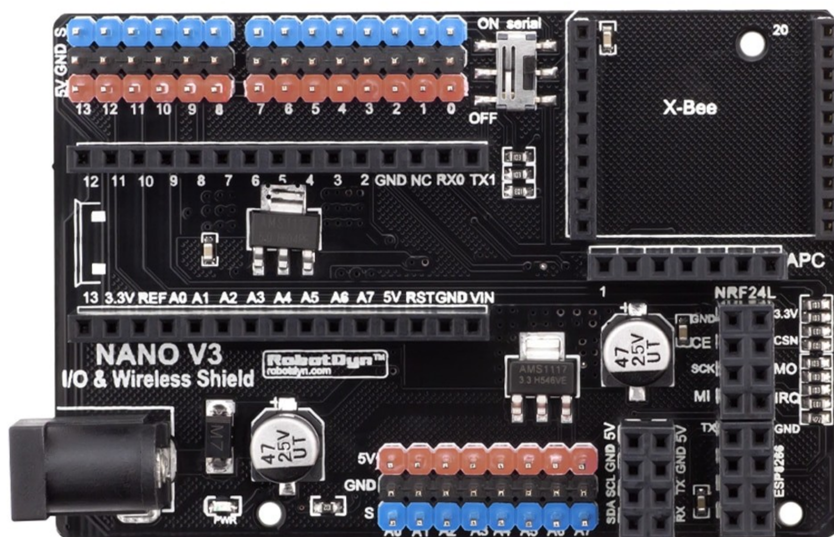
Aquesta placa permet controlar tires de leds RGB WS2812 amb el pin GPIO2. Es pot alimentar amb 3,7 V : 5 V, admeten bateries de Li de 3,7 V.





Nano V3.0 I/O & Wireless Shield

RobotDyn fabrica aquesta placa que admet un ESP-01, que només interacciona amb la comunicació sèrie de l’Arduino nano de la placa. Suficient per esclavitzar l’arduino nano i aprofitar la potència WiFi de l’ESP8266.



Part XI. Altres components externs interessants

Al llarg d’aquests anys he trobat un bon grapat de dispositius que m’han sorprès i agradat per mil motius: potència, preu, mida ... Voldria compartir amb vosaltres una selecció[43]:

Sensors

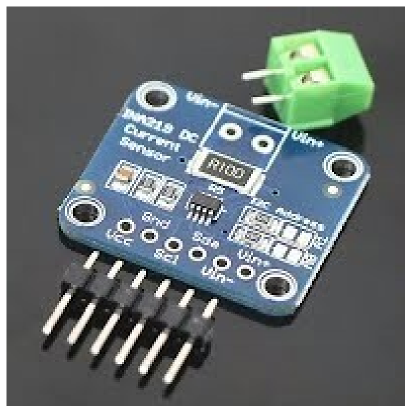
Brúixola HMC5883L

Protocol: I2C
Llibreries:
https://github.com/adafruit/Adafruit_Sensor + https://github.com/adafruit/Adafruit_HMC5883_Unified
Referències:
https://learn.adafruit.com/adafruit-hmc5883l-breakout-triple-axis-magnetometer-compass-sensor



Intensitat, tensió i potència: INA219

Protocol: I2C
Llibreria:
https://github.com/adafruit/Adafruit_INA219
Referències:
https://learn.adafruit.com/adafruit-ina219-current-sensor-breakout



43 Trobareu una col·lecció més extensa de dispositius a https://sites.google.com/a/iepegasoviana.cat/sensors-arduino/

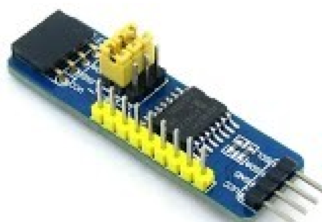
Mòdul CD74HC4067: multiplexor analògic 16 <-> 1

Protocol: I2C
Referències:
http://blog.codebender.cc/2014/01/30/mux74hc4067/



Mòdul PCF8574 I2C I/O

Protocol: I2C
Llibreria:
http://playground.arduino.cc/Main/PCF8574Class
Referències:
http://tronixstuff.com/2010/10/29/tutorial-arduino-and-the-i2c-bus-part-two/



Actuadors

Motor pas a pas: 28BYJ-48

Protocol: 4 sortides digitals
Referències:
https://sites.google.com/a/iepegasoviana.cat/sensors-arduino/actuadors/motor-pas-a-pas-28byj-48



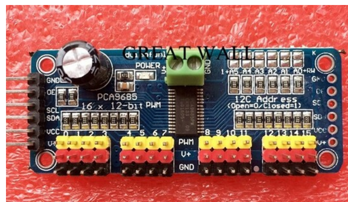
Mòdul L9110S pont H

Protocol: 4 sortides digitals
Referències:
https://www.bananarobotics.com/shop/How-to-use-the-HG7881-(L9110)-Dual-Channel-Motor-Driver-Module

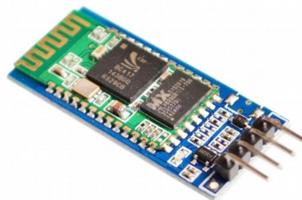
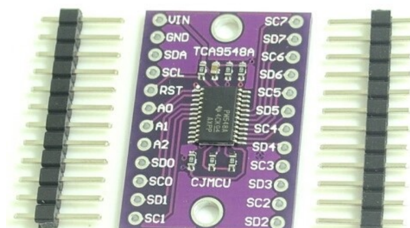


Mòdul PCA9685 I2C
16 servos / PWM

Protocol: I2C
Llibreria:
https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library
Referències:
https://learn.adafruit.com/16-channel-pwm-servo-driver



Comunicacions

Bluetooth slave: HC-06 Protocol: sèrie 3,3 V Referències: https://tronixlabs.com.au/news/tutorial-using-hc06-bluetooth-to-serial-wireless-uart-adaptors-with-arduino/	
I2C adaptador de nivells lògics Referències: https://www.sparkfun.com/products/12009	
I2C: multiplexor TCA9548A Protocol: I2C Referències: https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/wiring-and-test?view=all	

Part XII. Wearables

El nostre D1 mini és petit i lleuger, perfecte per fer projectes cosits a la roba.

Si bé podem utilitzar els nostres shields, fem una ullada als dispositius existents dissenyats expressament per a aquests projectes, on fins-i-tot trobarem arduinos que esclavitzar als nostres D1 mini:

LilyPad LED Light Module



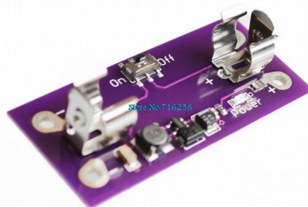
LilyPad Pixel Board WS2812 module



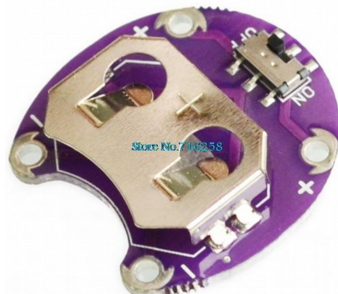
LilyPad Buzzer module



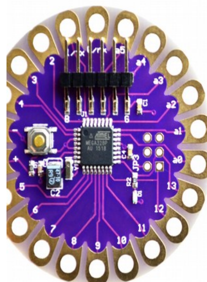
LilyPad Power Supply Module AAA Battery Step up to 5V Converter



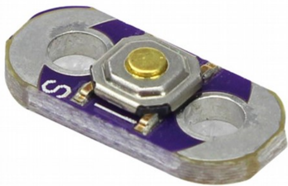
LilyPad Coin Cell Battery Holder CR2032 Battery Mount Module



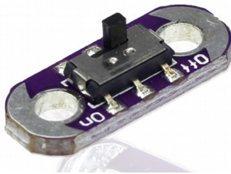
LilyPad 328 Main Board ATmega328P



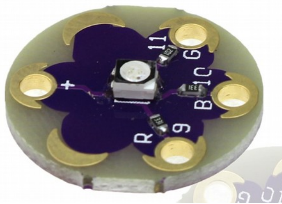
LilyPad Button Board Module



LilyPad Slide Switch



LilyPad Tri-Color LED RGB Module



Annexos

Relació d'annexos

Annex 1: Microcontrolador ESP8266

L’ESP8266 és un xip de baix cost SoC amb les següents prestacions : microprocessador, pila TCP/IP completa, connectivitat Wi-Fi i múltiples ports I/O disponibles, dissenyat per l'empresa xinesa Espressif Systems.

Veiem les seves característiques principals

- 32 bit CPU @ 80 MHz
- 64 kB command RAM
- 96 kB data RAM
- QPI Flash externa fins 16 MB
- WiFi 802.AA B/G/N 2,4 GHz, WEP/WPA/WPA2
- Fins a 16 pins I/O
- SPI, I2C, I2S, UART
- ADC de 10 bits

Annex 2: Altres llenguatges de programació per l'ESP8266

Treballar amb l'entorn Arduino no és la única opció que tenim per programa el D1 mini. Veiem algunes alternatives:

NodeMCU

Basat en el llenguatge de scripts Lua i creat expressament per l'ESP8266, concretament el mòdul ESP12 V1 (NodeMCU 0.9) que comentem al capítol Els mòduls ESP12 V1 i V3. La placa NODEMCU Base v1.0 .

Personalment el trobo més complicat que el C++ de l'arduino, tal vegada per no ser tant familiar.

Més informació a https://nodemcu.readthedocs.io/en/master/

Micropython

Implementació del llenguatge de programació Python 3 optimitzada per executar-se en un microcontrolador. Per tant parlem d'un llenguatge d'alt nivell molt potent i fàcilment llegible.

Trobareu més informació a https://micropython-on-wemos-d1-mini.readthedocs.io/en/latest/, on fa servir el D1 mini com a plataforma de desenvolupament.

MicroBlocks

Llenguatge gràfic inspirat en Scratch[44] que permet una programació dinàmica, paral·lela i interactiva. Actualment es troba en desenvolupament (versió Alpha) .

44 De fet els creadors de uBlocks són John Maloney, desenvolupador lider de Scrath durant 11 anys, Jens Mönig, programador lider de Snap, i Bernat Romagosa, autor de Snap4Arduino.

Personalment crec que aquest llenguatge té un gran futur i permetrà utilitzar el nostre kit D1 mini fins-i-tot a primària. Temps al temps.

Disponible a http://microblocks.fun/

BASIC

Inspirat en el treball de Carl Gundle[45], podem utilitzar aquest senzill llenguatge amb una interfície web amb editor, gestor de fitxers ... Molt complet i potent.

Disponible a https://www.esp8266basic.com/

45 Vegeu http://runbasic.com/

Annex 3: Connexions i compatibilitat dels shields

Shield	**D0**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**A0**
OLED		I2C	I2C	x	x					
Matrix led						x		x		
RTC		I2C	I2C							
Buzzer						x				
1 button				x						
RGB									x	
Relay								x		
DHT		I2C	I2C							
IR				x	x					
PIR							x			
CON1	x					x	x			
CON2		I2C	I2C							x

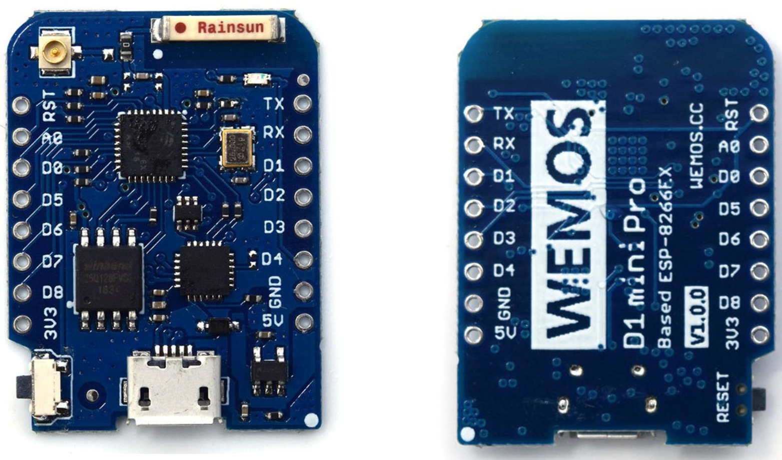
Annex 4: D1 mini pro

Diversos fabricants comercialitzen versions millorades del D1 mini amb el nom de D1 mini pro, si bé les seves prestacions i factor de forma poden ser ben diferents.

El propi fabricant Wemos / Lolin n’ha fet 3 versions diferents.

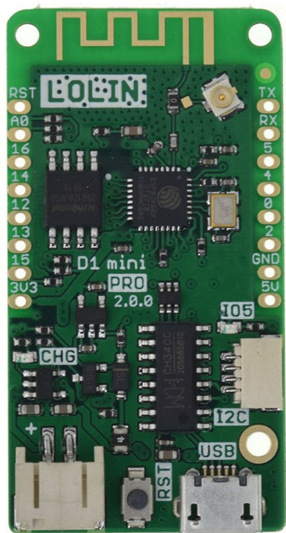
Les versions V1.0.0. i V1.1.0 són molts semblants, amb el factor de forma de la D1 estàndard. Únicament he trobat diferències mínimes (han posat un fusible de 0,5A a 5V, canvis d’alguns condensadors ...)

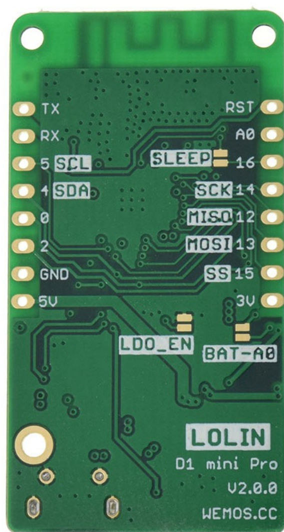
Totes dues porten integrada una antena ceràmica i un connector per a una antena exterior, i una memòria de 16 MB.



A la versió V2.0.0. han tornat a l’antena integrada al PCB, però mantenen el connector per a antena exterior. Han abandonat el factor de forma de la D1 estàndard, amb una placa més allargada, encara que compatible amb el bus de la D1 minio estàndard i les seves bases doble i triple. Han afegit un connector PH-2W0 i el circuit de

càrrega per a una bateria de Li, així com un connector I2C. La memòria també és de 16 MB.

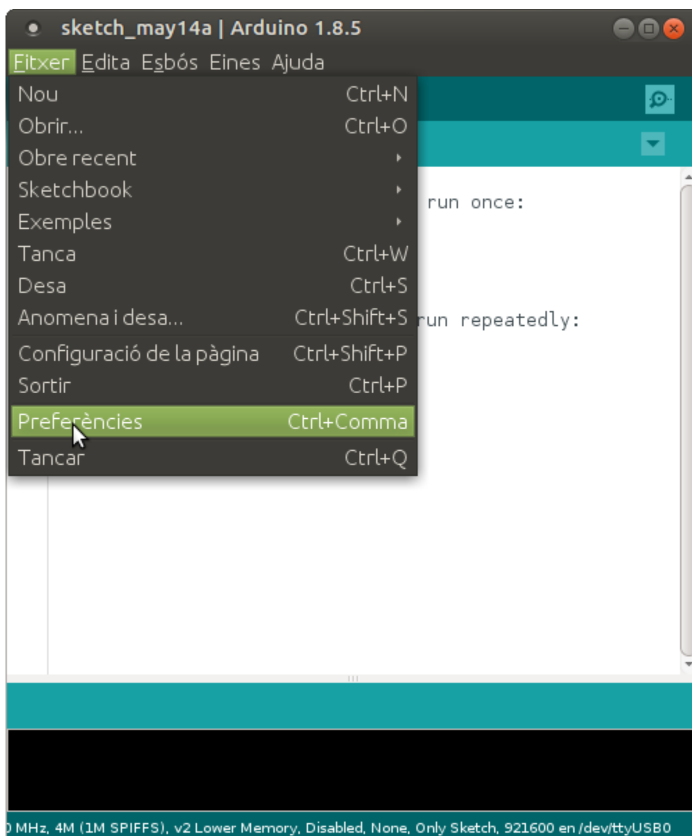




Annex 5: Instal·lació Arduino per a ESP8266

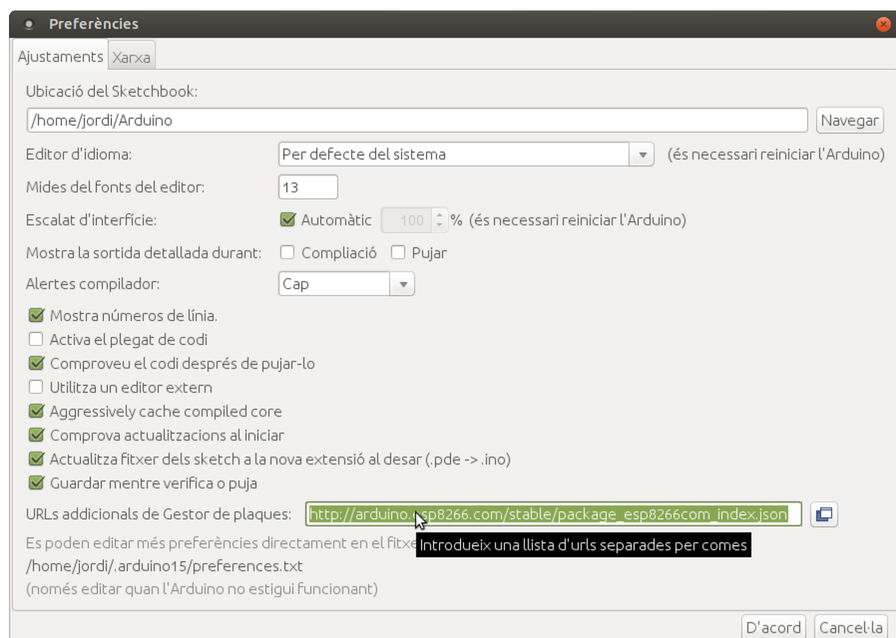
Ens caldrà un arduino IDE modern que permeti l'extensió a nous xips amb el Gestor targes. Nosaltres actualment fem servir la versió 1.8.5, però també hem fet servir la versió 1.6.8 sense cap problema.

Una vegada instal·lat, hem d'anar a *Fitxer->Preferències*

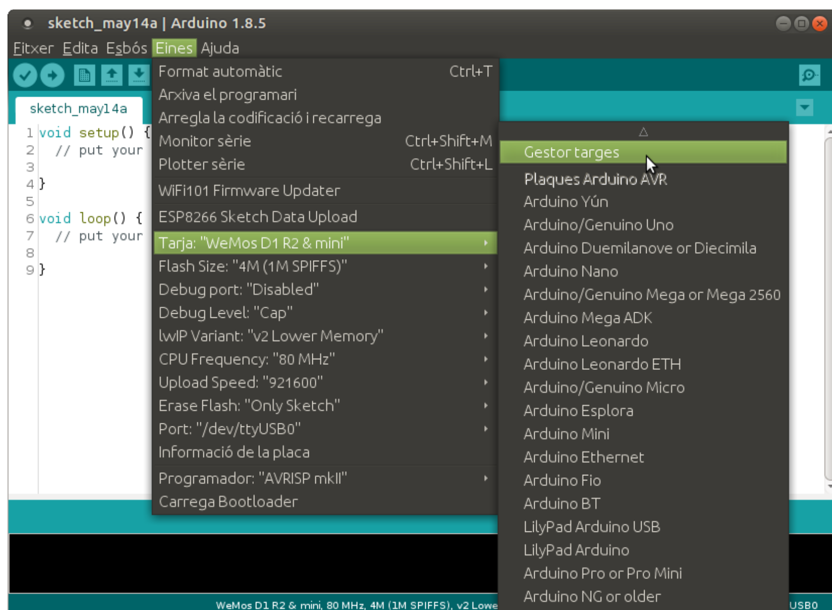


I omplir l'opció URLs adiccionals de Gestors de plaques amb

http://arduino.esp8266.com/stable/package_esp8266com_index.json

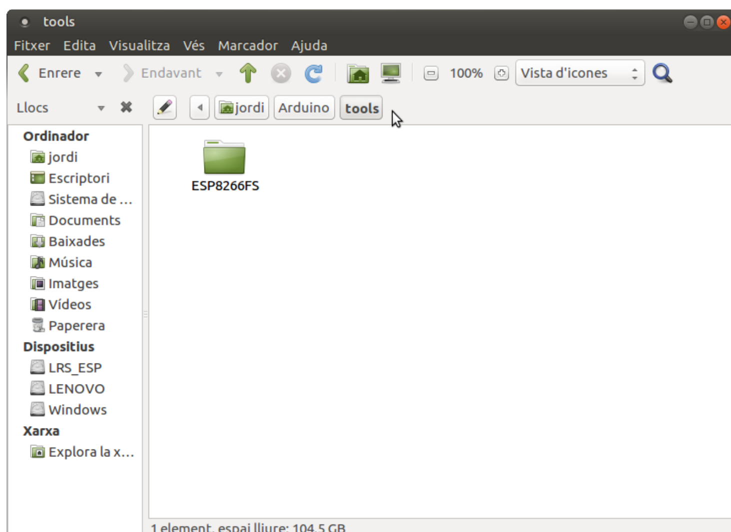


Ara podem anar a *Eines-> Tarja -> Gestor targes* i escollir *esp8266* per instal·lar-la





Finalment baixem l'arxiu https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.3.0/ESP8266FS-0.3.0.zip i el descomprimiu a la carpeta tools (potser haureu de crear aquesta carpeta) de la carpeta Arduino del vostre espai personal



Podeu trobar més informació a https://github.com/esp8266/Arduino

Annex 6: Taula de colors RGB

Aquest array el faig servir al meu exemple Peana per canviar de colors un led RGB amb la funció *setPixelColor()*:

```
uint32_t color[]={
  0xFFC0CB, //   0 Pink
  0xFFB6C1, //   1 LightPink
  0xFF69B4, //   2 HotPink
  0xFF1493, //   3 DeepPink
  0xDB7093, //   4 PaleVioletRed
  0xC71585, //   5 MediumVioletRed
  0xFFA07A, //   6 LightSalmon
  0xFA8072, //   7 Salmon
  0xE9967A, //   8 DarkSalmon
  0xF08080, //   9 LightCoral
  0xCD5C5C, //  10 IndianRed
  0xDC143C, //  11 Crimson
  0xB22222, //  12 FireBrick
  0x8B0000, //  13 DarkRed
  0xFF0000, //  14 Red
  0xFF4500, //  15 OrangeRed
  0xFF6347, //  16 Tomato
  0xFF7F50, //  17 Coral
  0xFF8C00, //  18 DarkOrange
  0xFFA500, //  19 Orange
  0xFFD700, //  20 Gold
  0xFFFF00, //  21 Yellow
  0xFFFFE0, //  22 LightYellow
  0xFFFACD, //  23 LemonChiffon
  0xFAFAD2, //  24 LightGoldenrodYellow
  0xFFEFD5, //  25 PapayaWhip
  0xFFE4B5, //  26 Moccasin
  0xFFDAB9, //  27 PeachPuff
  0xEEE8AA, //  28 PaleGoldenrod
  0xF0E68C, //  29 Khaki
  0xBDB76B, //  30 DarkKhaki
  0xFFF8DC, //  31 Cornsilk
  0xFFEBCD, //  32 BlanchedAlmond
  0xFFE4C4, //  33 Bisque
  0xFFDEAD, //  34 NavajoWhite
  0xF5DEB3, //  35 Wheat
  0xDEB887, //  36 BurlyWood
  0xD2B48C, //  37 Tan
  0xBC8F8F, //  38 RosyBrown
  0xF4A460, //  39 SandyBrown
  0xDAA520, //  40 Goldenrod
  0xB8860B, //  41 DarkGoldenrod
  0xCD853F, //  42 Peru
  0xD2691E, //  43 Chocolate
  0x8B4513, //  44 SaddleBrown
  0xA0522D, //  45 Sienna
  0xA52A2A, //  46 Brown
  0x800000, //  47 Maroon
  0x556B2F, //  48 DarkOliveGreen
  0x808000, //  49 Olive
  0x6B8E23, //  50 OliveDrab
  0x9ACD32, //  51 YellowGreen
  0x32CD32, //  52 LimeGreen
  0x00FF00, //  53 Lime
  0x7CFC00, //  54 LawnGreen
```

```
  0x7FFF00, //  55 Chartreuse
  0xADFF2F, //  56 GreenYellow
  0x00FF7F, //  57 SpringGreen
  0x00FA9A, //  58 MediumSpringGreen
  0x90EE90, //  59 LightGreen
  0x98FB98, //  60 PaleGreen
  0x8FBC8F, //  61 DarkSeaGreen
  0x3CB371, //  62 MediumSeaGreen
  0x2E8B57, //  63 SeaGreen
  0x228B22, //  64 ForestGreen
  0x008000, //  65 Green
  0x006400, //  66 DarkGreen
  0x66CDAA, //  67 MediumAquamarine
  0x00FFFF, //  68 Aqua
  0x00FFFF, //  69 Cyan
  0xE0FFFF, //  70 LightCyan
  0xAFEEEE, //  71 PaleTurquoise
  0x7FFFD4, //  72 Aquamarine
  0x40E0D0, //  73 Turquoise
  0x48D1CC, //  74 MediumTurquoise
  0x00CED1, //  75 DarkTurquoise
  0x20B2AA, //  76 LightSeaGreen
  0x5F9EA0, //  77 CadetBlue
  0x008B8B, //  78 DarkCyan
  0x008080, //  79 Teal
  0xB0C4DE, //  80 LightSteelBlue
  0xB0E0E6, //  81 PowderBlue
  0xADD8E6, //  82 LightBlue
  0x87CEEB, //  83 SkyBlue
  0x87CEFA, //  84 LightSkyBlue
  0x00BFFF, //  85 DeepSkyBlue
  0x1E90FF, //  86 DodgerBlue
  0x6495ED, //  87 CornflowerBlue
  0x4682B4, //  88 SteelBlue
  0x4169E1, //  89 RoyalBlue
  0x0000FF, //  90 Blue
  0x0000CD, //  91 MediumBlue
  0x00008B, //  92 DarkBlue
  0x000080, //  93 Navy
  0x191970, //  94 MidnightBlue
  0xE6E6FA, //  95 Lavender
  0xD8BFD8, //  96 Thistle
  0xDDA0DD, //  97 Plum
  0xEE82EE, //  98 Violet
  0xDA70D6, //  99 Orchid
  0xFF00FF, // 100 Magenta
  0xBA55D3, // 101 MediumOrchid
  0x9370DB, // 102 MediumPurple
  0x8A2BE2, // 103 BlueViolet
  0x9400D3, // 104 DarkViolet
  0x9932CC, // 105 DarkOrchid
  0x8B008B, // 106 DarkMagenta
  0x800080, // 107 Purple
  0x4B0082, // 108 Indigo
  0x483D8B, // 109 DarkSlateBlue
  0x6A5ACD, // 110 SlateBlue
  0x7B68EE, // 111 MediumSlateBlue
  0xFFFFFF, // 112 White
  0xF5F5DC, // 113 Beige
  0xFAEBD7, // 114 AntiqueWhite
  0xFFE4E1, // 115 MistyRose
  0x808080, // 116 Gray
  0x708090  // 117 SlateGray
};
```

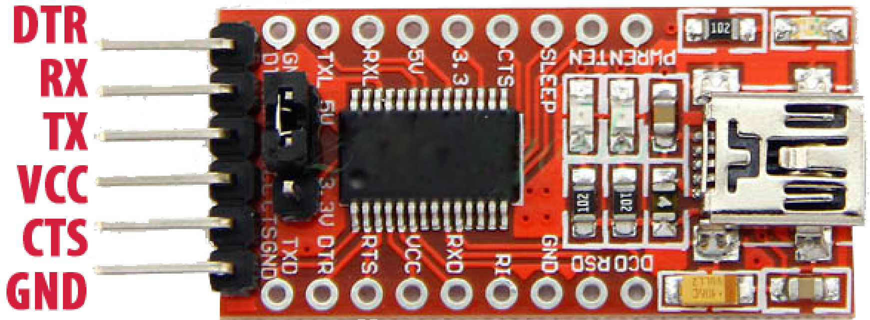
Annex 7: Taula de freqüències per a les notes musicals

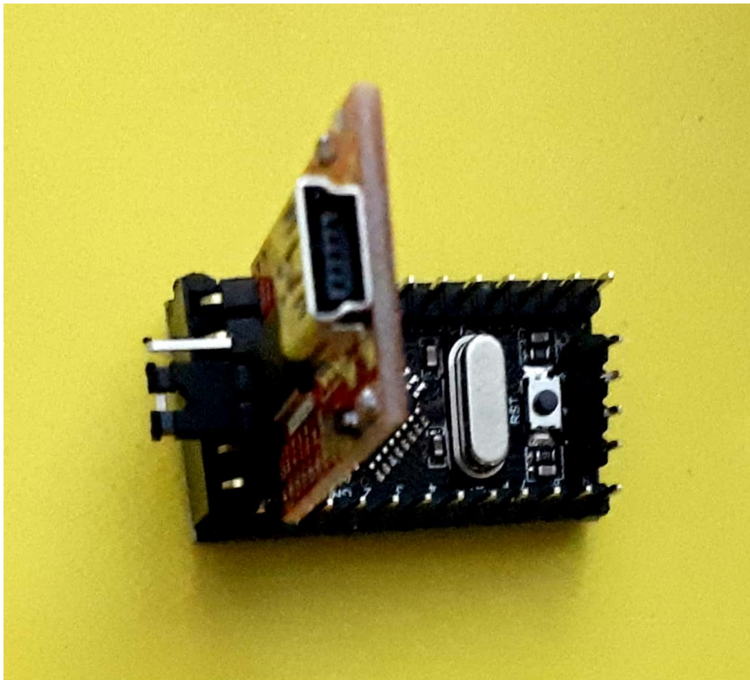
Aquest array el faig servir al meu exemple musica:

```
int notes [36][3] =
{
{'8',B00100000,262}, // C5
{'(',B00100001,277}, // C5#
{'9',B00100010,294}, // D5
{')',B00100011,311}, // D5#
{'0',B00100100,330}, // E5
{'q',B00100101,349}, // F5
{'Q',B00100110,370}, // F5#
{'w',B00100111,392}, // G5
{'W',B00101000,415}, // G5#
{'e',B00101001,440}, // A5
{'E',B00101010,466}, // A5#
{'r',B00101011,494}, // B5
{'t',B00000000,523}, // C6
{'T',B00000001,554}, // C6#
{'y',B00000010,587}, // D6
{'Y',B00000011,622}, // D6#
{'u',B00000100,669}, // E6
{'i',B00000101,698}, // F6
{'I',B00000110,740}, // F6#
{'o',B00000111,784}, // G6
{'O',B00001000,831}, // G6#
{'p',B00001001,880}, // A6
{'P',B00001010,932}, // A6#
{'a',B00001011,988}, // B6
{'s',B00010000,1047}, // C7
{'S',B00010001,1109}, // C7#
{'d',B00010010,1175}, // D7
{'D',B00010011,1245}, // D7#
{'f',B00010100,1318}, // E7
{'g',B00010101,1396}, // F7
{'G',B00010110,1480}, // F7#
{'h',B00010111,1568}, // G7
{'H',B00011000,1661}, // G7#
{'j',B00011001,1760}, // A7
{'J',B00011010,1865}, // A7#
{'k',B00011011,1975}, // B7
};
```

Annex 8: Firmware de l’arduino pro mini

Podem utilitzar un mòdul FTDI232 endollant-lo directament a l’arduino pro mini per carregar el firmware:





I2CsapmR1_firmware.h (V1.0)

```
#define I2CsapmR1_A0    0
#define I2CsapmR1_A1    1
#define I2CsapmR1_A2    2
#define I2CsapmR1_A3    3
#define I2CsapmR1_A6    4
#define I2CsapmR1_A7    5
#define I2CsapmR1_D2    6
#define I2CsapmR1_D3    7
#define I2CsapmR1_D4    8
#define I2CsapmR1_D5    9
#define I2CsapmR1_D6   10
#define I2CsapmR1_D7   11
#define I2CsapmR1_D8   12
#define I2CsapmR1_D9   13
#define I2CsapmR1_D10  14
#define I2CsapmR1_D11  15
#define I2CsapmR1_D12  16
#define I2CsapmR1_D13  17
#define I2CsapmR1_M2   18
#define I2CsapmR1_M3   19
#define I2CsapmR1_M4   20
#define I2CsapmR1_M5   21
#define I2CsapmR1_M6   22
#define I2CsapmR1_M7   23
#define I2CsapmR1_M8   24
#define I2CsapmR1_M9   25
#define I2CsapmR1_M10  26
#define I2CsapmR1_M11  27
#define I2CsapmR1_M12  28
#define I2CsapmR1_M13  29
```

I2CsapmR1_firmware.cpp (V1.0)

```
#include "I2CsapmR1_firmware.h"
#include <Wire.h>
#define I2CADDR 2

 byte ordre = 0;
 byte dada = 0;

 int analin[6];
 int digm[12];
 int digd[12];

 int i;

void setup() {
  pinMode(2,INPUT);
  pinMode(3,INPUT);
  pinMode(4,INPUT);
  pinMode(5,INPUT);
  pinMode(6,INPUT);
  pinMode(7,INPUT);
  pinMode(8,INPUT);
  pinMode(9,INPUT);
  pinMode(10,INPUT);
  pinMode(11,INPUT);
  pinMode(12,INPUT);
  pinMode(13,INPUT);
  for (i=0;i<12;i++) digm[i]=INPUT;
  Wire.begin(I2CADDR);
```

```
  Wire.onReceive(escolta);
  Wire.onRequest(respon);
}

void loop() {
    analin[0]=analogRead(A0);
    analin[1]=analogRead(A1);
    analin[2]=analogRead(A2);
    analin[3]=analogRead(A3);
    analin[4]=analogRead(A6);
    analin[5]=analogRead(A7);
    for (i=0;i<12;i++){
      if ((digm[i]==INPUT) || (digm[i]==INPUT_PULLUP)){
        digd[i]=digitalRead(i+2);
      }

    }

}

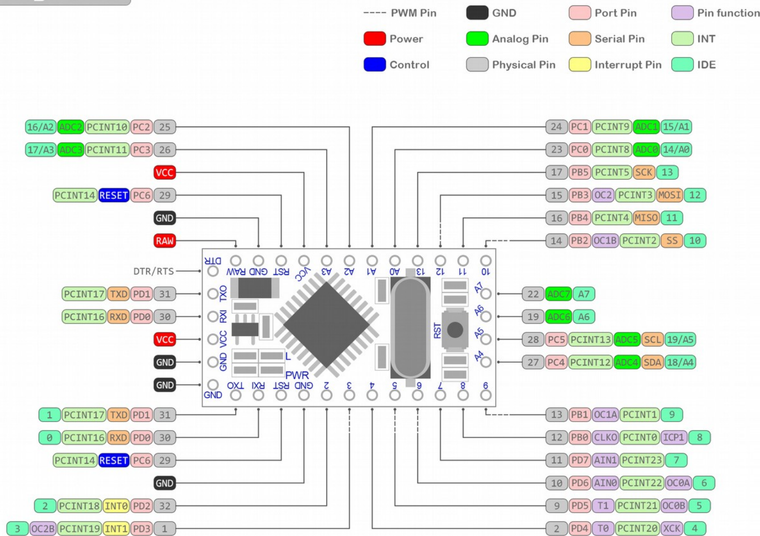
void escolta(int ordrelen) {
  ordre = Wire.read();
  if (ordrelen>1) {
  dada =  Wire.read();
  switch(ordre) {
    case I2CsapmR1_M2:
      pinMode(2,dada);
      break;
    case I2CsapmR1_M3:
      pinMode(3,dada);
      break;
    case I2CsapmR1_M4:
      pinMode(4,dada);
      break;
    case I2CsapmR1_M5:
      pinMode(5,dada);
      break;
    case I2CsapmR1_M6:
      pinMode(6,dada);
      break;
    case I2CsapmR1_M7:
      pinMode(7,dada);
      break;
    case I2CsapmR1_M8:
      pinMode(8,dada);
      break;
    case I2CsapmR1_M9:
      pinMode(9,dada);
      break;
    case I2CsapmR1_M10:
      pinMode(10,dada);
      break;
    case I2CsapmR1_M11:
      pinMode(11,dada);
      break;
    case I2CsapmR1_M12:
      pinMode(12,dada);
      break;
    case I2CsapmR1_M13:
      pinMode(13,dada);
      break;
    case I2CsapmR1_D2:
      digitalWrite(2,dada);
      break;
```

```
    case I2CsapmR1_D3:
      digitalWrite(3,dada);
      break;
    case I2CsapmR1_D4:
      digitalWrite(4,dada);
      break;
    case I2CsapmR1_D5:
      digitalWrite(5,dada);
      break;
    case I2CsapmR1_D6:
      digitalWrite(6,dada);
      break;
    case I2CsapmR1_D7:
      digitalWrite(7,dada);
      break;
    case I2CsapmR1_D8:
      digitalWrite(8,dada);
      break;
    case I2CsapmR1_D9:
      digitalWrite(9,dada);
      break;
    case I2CsapmR1_D10:
      digitalWrite(10,dada);
      break;
    case I2CsapmR1_D11:
      digitalWrite(11,dada);
      break;
    case I2CsapmR1_D12:
      digitalWrite(12,dada);
      break;
    case I2CsapmR1_D13:
      digitalWrite(13,dada);
      break;
    }
  }
}

void respon() {
  int resposta;
  switch(ordre) {
    case I2CsapmR1_A0:
      resposta = analin[0];
      break;
    case I2CsapmR1_A1:
      resposta = analin[1];
      break;
    case I2CsapmR1_A2:
      resposta = analin[2];
      break;
    case I2CsapmR1_A3:
      resposta = analin[3];
      break;
    case I2CsapmR1_A6:
      resposta = analin[4];
      break;
    case I2CsapmR1_A7:
      resposta = analin[5];
      break;
    case I2CsapmR1_D2:
      resposta = digd[0];
      break;
    case I2CsapmR1_D3:
      resposta = digd[1];
      break;
```

```
    case I2CsapmR1_D4:
      resposta = digd[2];
      break;
    case I2CsapmR1_D5:
      resposta = digd[3];
      break;
    case I2CsapmR1_D6:
      resposta = digd[4];
      break;
    case I2CsapmR1_D7:
      resposta = digd[5];
      break;
    case I2CsapmR1_D8:
      resposta = digd[6];
      break;
    case I2CsapmR1_D9:
      resposta = digd[7];
      break;
    case I2CsapmR1_D10:
      resposta = digd[8];
      break;
    case I2CsapmR1_D11:
      resposta = digd[9];
      break;
    case I2CsapmR1_D12:
      resposta = digd[10];
      break;
    case I2CsapmR1_D13:
      resposta = digd[11];
      break;
    default:
      resposta = -1;
      break;
  }
  byte buffer[2];
  buffer[0]=resposta >> 8;
  buffer[1]=resposta & 0xff;
  Wire.write(buffer,2);
  ordre=0;
}
```

Podem adaptar aquest firmware a les nostres necessitats. Les possibilitats són infinites!


RobotDyn®
www.robotdyn.com
PINOUT DIAGRAM
ProMini
ATmega328P
PWM Pin
GND
Port Pin
Pin function
Power
Analog Pin
Serial Pin
INT
Control
Physical Pin
Interrupt Pin
IDE
RobotDyn®
04 Aug 2017

Annex 9: Afegint la funcionalitat de FSManager als nostres projectes

Com vau veure a Incloure eines HTML5 a la SD virtual si voleu afegir FSManager al vostre programa només cal copiar els arxius FSManager.cpp i FSManager.h a la carpeta del vostre programa i afegir al principi les línies:

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;
```

i al *setup()*

```
  SPIFFS.begin();
  initHelper(); //inclou connexió Wifi i server
                // Aquí podem afegir altres funcions server.on()
  server.begin();
```

I fer servir la funció *espera()* al *loop()* en lloc de *delay()* per poder atendre les peticions web.

Recordeu canviar a l’inici de FSManager.cpp el nom de la vostre xarxa i contrasenya i, si s’escau, el mode WiFi.

Contingut de *FSManager.h*:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#define DBG_OUTPUT_PORT Serial
#include <FS.h>

// Helper functions prototypes
void OkRetorn();
void printDirectory();
void ajaxDirectory();
void handleFileCreate();
void handleFileDelete();
void handleFileUpload();
String formatBytes(size_t bytes);
String getContentType(String filename);
bool handleFileRead(String path);
void handleFileEdit();
void handleFileSave();
void initHelper();
void initWifi();
void pageHead();
void espera(int temps);
```

Contingut de *FSManager.cpp*:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#define DBG_OUTPUT_PORT Serial
#include <FS.h>

// Helper functions prototypes
void OkRetorn();
void printDirectory();
void ajaxDirectory();
void handleFileCreate();
void handleFileDelete();
void handleFileUpload();
String formatBytes(size_t bytes);
String getContentType(String filename);
bool handleFileRead(String path);
void handleFileEdit();
void handleFileSave();
void initHelper();
void initWifi();
void pageHead();
void espera(int temps);

const char* ssid = "nom de la xarxa";
const char* password = "contrasenya";
const char* host = "esp8266fs";
// tria el tipus de connexió. Client WIFI_STA, Access Point WIFI_AP
#define MODE_WIFI WIFI_STA

ESP8266WebServer server(80);
File fsUploadFile;

void espera(int temps) {
  unsigned long ara = millis();
  unsigned long seguent = ara + temps;
  delay(1); //refresh watchdog
  while (millis()<seguent){
      server.handleClient();
  }
}

void pageHead(){
  server.setContentLength(CONTENT_LENGTH_UNKNOWN);
  server.send(200, "text/html", "");
  server.sendContent("<!DOCTYPE html><html><head><title>Gesti&oacute; de 
fitxersr</title><meta charset='UTF-8'><meta name='viewport' 
content='width=device-width'/></head><body>");
}

void initWifi(){
  DBG_OUTPUT_PORT.begin(115200);
  if (MODE_WIFI==WIFI_STA) {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    DBG_OUTPUT_PORT.printf("Connecting to %s\n", ssid);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      DBG_OUTPUT_PORT.print(".");
    }
    DBG_OUTPUT_PORT.println("");
    DBG_OUTPUT_PORT.print("Connected! IP address: ");
```

```
    DBG_OUTPUT_PORT.println(WiFi.localIP());

    DBG_OUTPUT_PORT.print("Open http://");
    DBG_OUTPUT_PORT.print(WiFi.localIP());
    DBG_OUTPUT_PORT.println("/dir to see the file browser");
  }
  else {
    WiFi.mode(WIFI_AP);
    WiFi.softAP(ssid, password);
    DBG_OUTPUT_PORT.print("Open http://192.168.4.1/dir to see the file
browser");
  }
}

void initHelper(){
  DBG_OUTPUT_PORT.print("\n");
  DBG_OUTPUT_PORT.setDebugOutput(true);
    Dir dir = SPIFFS.openDir("/");
    while (dir.next()) {
      String fileName = dir.fileName();
      size_t fileSize = dir.fileSize();
      DBG_OUTPUT_PORT.printf("FS File: %s, size: %s\n", fileName.c_str(),
formatBytes(fileSize).c_str());
      DBG_OUTPUT_PORT.printf("\n");
  }

  initWifi();

  //SERVER INIT
  server.on("/dir", HTTP_GET, printDirectory);
  server.on("/ajaxdir", HTTP_GET, ajaxDirectory);
  server.on("/create", HTTP_GET, handleFileCreate);
  server.on("/delete", HTTP_GET, handleFileDelete);
  //first callback is called after the request has ended with all parsed
arguments
  //second callback handles file uploads at that location
  server.on("/upload", HTTP_POST, []() {
      OkRetorn();
  }, handleFileUpload);
  server.on("/edit", HTTP_GET, handleFileEdit);
  server.on("/save", HTTP_POST, handleFileSave);
  //called when the url is not defined here
  //use it to load content from SPIFFS
  server.onNotFound([]() {
    if (!handleFileRead(server.uri())) {
      server.send(404, "text/plain", "FileNotFound");
    }
  });
//  server.begin();


}

void OkRetorn() {
  pageHead();
  server.sendContent("Operaci&oacute; realitzada amb &egrave;xit<br/><a
href='/dir'>Tornar a gesti&oacute; de fitxers</a></body></html>");
}

void printDirectory(){
  Dir dir = SPIFFS.openDir("/");
  pageHead();
  server.sendContent("<h1>Gesti&oacute; de fitxers SD virtual</h1>");
  while (dir.next()) {
```

```
    String fileName = dir.fileName();
    size_t fileSize = dir.fileSize();
    DBG_OUTPUT_PORT.printf("FS File: %s, size: %s\n", fileName.c_str(),
formatBytes(fileSize).c_str());
    String output;
    output += "<a href='/";
    output += fileName.substring(1);
    output +="'>";
    output += fileName.substring(1);
    output +="</a> ";
    output += formatBytes(fileSize).c_str();
    output += "      <a href='";
    output += fileName;
    output +="?download=1' style='text-decoration: none;'> &#9921;
</a>  ";
    output += "      <a href='/edit?fe=";
    output += fileName;
    output +="' style='text-decoration: none;'> &#9997; </a>  ";
    output += "      <a href='/delete?killfitxer=";
    output += fileName;
    output +="' style='text-decoration: none;'> &#10006; </a><br/>";
    server.sendContent(output);
    DBG_OUTPUT_PORT.println(output);

  }
  FSInfo fs_info;
  SPIFFS.info(fs_info);
  String output;
  output += formatBytes(fs_info.usedBytes).c_str();
  output +=" ocupats de ";
  output += formatBytes(fs_info.totalBytes).c_str();
  output +=" totals";
  output +="<br/><br/>";
  server.sendContent(output);

  server.sendContent("<br/><form action='/create'>Fitxer nou: <input
type='text' name='noufitxer' value=''>  <input type='submit'
value='Crea'></form>");
  server.sendContent("<br/><form method='post' enctype='multipart/form-data'
action='/upload'>Fitxer a enviar: <input type='file' name='myFile'><input
type='submit' value='Envia'></form>");
  server.sendContent("</body></html>");
}

void ajaxDirectory(){
  server.setContentLength(CONTENT_LENGTH_UNKNOWN);
  server.send(200, "text/plain", "");
  Dir dir = SPIFFS.openDir("/");
  while (dir.next()) {
    String fileName = dir.fileName();
    String output;
    output += fileName;
    output +=",";
    server.sendContent(output);
  }
}


void handleFileCreate() {
  if (server.args() == 0) {
    return server.send(500, "text/plain", "BAD ARGS");
  }
  String path = "/" + server.arg("noufitxer");
  DBG_OUTPUT_PORT.println("handleFileCreate: " + path);
```

```
  if (path == "/") {
    return server.send(500, "text/plain", "BAD PATH");
  }
  if (SPIFFS.exists(path)) {
    return server.send(500, "text/plain", "FILE EXISTS");
  }
  File file = SPIFFS.open(path, "w");
  if (file) {
    file.close();
  } else {
    return server.send(500, "text/plain", "CREATE FAILED");
  }
  OkRetorn();
  path = String();
}

void handleFileDelete() {
  if (server.args() == 0) {
    return server.send(500, "text/plain", "BAD ARGS");
  }
  String path = server.arg("killfitxer");
  DBG_OUTPUT_PORT.println("handleFileDelete: " + path);
  if (path == "/") {
    return server.send(500, "text/plain", "BAD PATH");
  }
  if (!SPIFFS.exists(path)) {
    return server.send(404, "text/plain", "FileNotFound");
  }
  SPIFFS.remove(path);
  OkRetorn();
  path = String();
}

void handleFileUpload() {
  if (server.uri() != "/upload") {
    return;
  }
  HTTPUpload& upload = server.upload();
  if (upload.status == UPLOAD_FILE_START) {
    String filename = upload.filename;
    if (!filename.startsWith("/")) {
      filename = "/" + filename;
    }
    DBG_OUTPUT_PORT.print("handleFileUpload Name: ");
DBG_OUTPUT_PORT.println(filename);
    fsUploadFile = SPIFFS.open(filename, "w");
    filename = String();
  } else if (upload.status == UPLOAD_FILE_WRITE) {
    //DBG_OUTPUT_PORT.print("handleFileUpload Data: ");
DBG_OUTPUT_PORT.println(upload.currentSize);
    if (fsUploadFile) {
      fsUploadFile.write(upload.buf, upload.currentSize);
    }
  } else if (upload.status == UPLOAD_FILE_END) {
    if (fsUploadFile) {
      fsUploadFile.close();
    }
    DBG_OUTPUT_PORT.print("handleFileUpload Size: ");
DBG_OUTPUT_PORT.println(upload.totalSize);
  }
}

String formatBytes(size_t bytes) {
  if (bytes < 1024) {
```

```
    return String(bytes) + "B";
  } else if (bytes < (1024 * 1024)) {
    return String(bytes / 1024.0) + "KB";
  } else if (bytes < (1024 * 1024 * 1024)) {
    return String(bytes / 1024.0 / 1024.0) + "MB";
  } else {
    return String(bytes / 1024.0 / 1024.0 / 1024.0) + "GB";
  }
}

String getContentType(String filename) {
  if (server.hasArg("download")) {
    return "application/octet-stream";
  } else if (filename.endsWith(".htm")) {
    return "text/html";
  } else if (filename.endsWith(".html")) {
    return "text/html";
  } else if (filename.endsWith(".css")) {
    return "text/css";
  } else if (filename.endsWith(".js")) {
    return "application/javascript";
  } else if (filename.endsWith(".png")) {
    return "image/png";
  } else if (filename.endsWith(".gif")) {
    return "image/gif";
  } else if (filename.endsWith(".jpg")) {
    return "image/jpeg";
  } else if (filename.endsWith(".svg")) {
    return "image/svg+xml";
  } else if (filename.endsWith(".ico")) {
    return "image/x-icon";
  } else if (filename.endsWith(".xml")) {
    return "text/xml";
  } else if (filename.endsWith(".pdf")) {
    return "application/x-pdf";
  } else if (filename.endsWith(".zip")) {
    return "application/x-zip";
  } else if (filename.endsWith(".gz")) {
    return "application/x-gzip";
  }
  return "text/plain";
}

bool handleFileRead(String path) {
  DBG_OUTPUT_PORT.println("handleFileRead: " + path);
  if (path.endsWith("/")) {
    path += "index.htm";
  }
  String contentType = getContentType(path);
  String pathWithGz = path + ".gz";
  if (SPIFFS.exists(pathWithGz) || SPIFFS.exists(path)) {
    if (SPIFFS.exists(pathWithGz)) {
      path += ".gz";
    }
    File file = SPIFFS.open(path, "r");
    server.streamFile(file, contentType);
    file.close();
    return true;
  }
  return false;
}

void handleFileEdit() {
  if (server.args() == 0) {
```

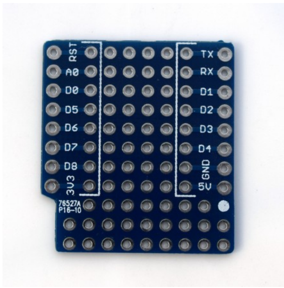
```
    return server.send(500, "text/plain", "BAD ARGS");
  }
  String path = server.arg("fe");
  DBG_OUTPUT_PORT.println("handleFileEdit: " + path);
  if (path == "/") {
    return server.send(500, "text/plain", "BAD PATH");
  }
  if (!SPIFFS.exists(path)) {
    DBG_OUTPUT_PORT.print("handleFileEdit Name: ");
DBG_OUTPUT_PORT.print(path);DBG_OUTPUT_PORT.println(" FileNotFound");
    return server.send(404, "text/plain", "FileNotFound");
  }
  pageHead();
  server.sendContent("<form action='/save' method='POST'>Fitxer: <input
type='text' name='nomfitxer' value='");
  server.sendContent(path);
  server.sendContent("' readonly><br/><textarea name='cos' rows='30' cols='40'
wrap='off'>");

  File file = SPIFFS.open(path,"r");
  char buffer[2];
  buffer[1] = 0;
//  server.streamFile(file, "text/plain");
  while (file.readBytes(buffer, 1)!=0 ) {
    server.sendContent(buffer);
  }
  file.close();
  server.sendContent("</textarea><br><input type='submit'
value='Desa'></form></body></html>");

  path = String();
}

void handleFileSave() {
  String path = server.arg("nomfitxer");
  String text = server.arg("cos");
  DBG_OUTPUT_PORT.println("handleFileSave: " + path);
  DBG_OUTPUT_PORT.println("text: " + text);
  File file = SPIFFS.open(path,"w");
  if (!file) {
    DBG_OUTPUT_PORT.println("file open failed");
  }
  file.print(text);
  file.close();
  OkRetorn();
  path = String();
  text = String();
}
```

Annex 10: Plantilla connexions projecte D1



Projecte:

Data:

Alumnes:

D1 mini	**GPIO**	**Shield / mòdul**	**Notes**
D0	16		
D1 / *SCL*	5		
D2 / *SDA*	4		
D3	0		pull-up 12 kΩ
D4	2		pull-up 12 kΩ *LED_BUILTIN*
D5	14		
D6	12		
D7	13		
D8	15		pull-down 12 kΩ
A0			Màx 3,3 V Z_i = 320 kΩ

Annex 11: Llistat del fitxer *OTA.cpp*

Al capítol Part IX. Actualització WiFi del programa (OTA) vam veure com podem actualitzar per WiFi el nostre programa. Aquí teniu el llistat del fitxer *OTA.cpp* que heu de copiar a la carpeta del vostre programa:

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>

const char* OTAssid = "nom de la xarxa";
const char* OTApassword = "contrasenya";

void OTAwait(){
  WiFi.mode(WIFI_STA);
  WiFi.begin(OTAssid, OTApassword);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) delay(1);
  // Port defaults to 8266
  // ArduinoOTA.setPort(8266);
  // Hostname defaults to esp8266-[ChipID]
  // ArduinoOTA.setHostname("myesp8266");
  // No authentication by default
  // ArduinoOTA.setPassword("admin");
  // Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  // ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");
  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH) {
      type = "sketch";
    } else { // U_SPIFFS
      type = "filesystem";
    }
    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using
SPIFFS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) {
      Serial.println("Auth Failed");
    } else if (error == OTA_BEGIN_ERROR) {
      Serial.println("Begin Failed");
    } else if (error == OTA_CONNECT_ERROR) {
      Serial.println("Connect Failed");
    } else if (error == OTA_RECEIVE_ERROR) {
      Serial.println("Receive Failed");
    } else if (error == OTA_END_ERROR) {
      Serial.println("End Failed");
    }
```

```
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  while(true) {
      ArduinoOTA.handle();
      delay(1);
  }
}
```

Bibliografia

[W3C01] Some early ideas for HTML. World Wide Web Consortium. [en línia] [consulta: 5 de gener de 2018]. Disponible a https://www.w3.org/MarkUp/historical

[IPV01] MAKER FAIRE BARCELONA 2018. INS Príncep de Viana. [en línia] [consulta: 5 de gener de 2018]. Disponible a https://agora.xtec.cat/iespviana/general/maker-faire-barcelona-2018/

[CES01] Jornada «Engrescant el jovent cap a la tecno». Centre de Recursos Pedagògics Específics de Suport a la Innovació i la Recerca Educativa (CESIRE). [en línia] [consulta: 5 de gener de 2018]. Disponible a https://agora.xtec.cat/cesire/general/jornada-engrescant-el-jovent-cap-a-la-tecno/

[GIT01] kit D1 mini. Jordi Orts. [en línia] [consulta: 5 de gener de 2018]. Disponible a https://github.com/jorts64/kit-D1-mini/wiki

[GIT02] kit D1 mini. Jordi Orts. [en línia] [consulta: 5 de gener de 2018]. Disponible a https://github.com/jorts64/kit-D1-mini/wiki/kit-D1-mini-R1

[EAC01] ESP8266 Arduino Core. ESP8266WIFI. Class Description. Station. [en línia] [consulta: 5 de gener de 2018]. Disponible a https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html#station